Using Visual Features to Generate Language Context

Undergraduate Honors Research Thesis

Presented in Partial Fulfillment of the Requirements for
Undergraduate Honors Research Thesis in the College of Engineering
at The Ohio State University

By

Isaac Zachmann,

Department of Electrical and Computer Engineering

The Ohio State University

2021

Dissertation Committee:

Dr. Jim Davis, OSU (CSE), Advisor

Dr. Tim Anderson, AFRL (WPAFB)

Jeremy Gwinnup, AFRL (WPAFB)

Dr. Emre Ertin, OSU (ECE)

Abstract

Current multimodal data processing methods use deep learning to combine complementary visual and textual information. Although large neural networks can be useful, they are often computationally intensive and require very large training datasets. This thesis explores a novel and direct method for predicting useful and understandable language context from the visual information within a video. Each video is represented by a scene-weighted sum of deep feature vectors across the video. Then the method predicts a novel cluster-weighted term frequency-inverse document frequency (TF–IDF) vector for a test video by averaging the TF–IDF vectors of the K training videos having the most similar visual features. The predicted vector provides information on what words are likely to be the most important in the video. We demonstrate that this method provides reliable textual information for a collection of instructional YouTube videos. The predicted TF–IDF vectors could be used to aid in speech-to-text or machine translation applications by providing a rich semantic context.

This work is dedicated to my friends and family.

Acknowledgments

I would like to thank Dr. Jim Davis for his support throughout this project. Dr. Davis has provided me with guidance throughout the last year. I am grateful for everything he has taught me throughout my work on this project. He has supported my ideas and worked with me to develop them. I would like to thank Tim Anderson and Air Force Research Laboratories for funding this project and for helpful guidance in the direction of the project. I would also like to thank Jeremy Gwinnup for his support and feedback.

I would like to thank all members of the Computer Vision Laboratory for creating an enjoyable research environment. I want to thank Logan for recommending me to join the lab. Jamie introduced me to the FINCH clustering algorithm that was used in this thesis. I would also like to thank Tong, Ash, and Nick for all the knowledge they shared during our weekly meetings. Lastly, I am thankful for all of my fellow buckeyes for giving me a great Ohio State experience.

Vita

Expected Spring 2022	Bachelor of Science Electrical and Computer Engineering The Ohio State University
Spring 2020-present	Undergraduate Research Assistant Computer Vision Lab The Ohio State University

Publications

Research Publications

I. Zachmann and T. Scarnati "A comparison of template matching and deep learning for classification of occluded targets in LiDAR data." *Automatic Target Recognition XXX.* Vol. 11394. International Society for Optics and Photonics, 2020.

T. Scarnati, R. Shaver, M. Saville, P. Sotirelis, R. Profetta, and I. Zachmann "Electro-Optical and Radio Frequency Data Fusion for Generating Three-Dimensional Point Clouds." *Military Sensing Symposium, Tri-Service Radar Symposium,* June 2019.

Fields of Study

Major Field: Electrical and Computer Engineering

Table of Contents

List of Tables

Table **Page**

List of Figures

Chapter 1: Introduction

Data can be gathered from the environment in many different ways. When multiple modes are used to gather data, such as listening with ears and seeing with eyes, the data is multimodal. Multimodal datasets can consist of visual information, acoustic information, textual information, and more. Sometimes information gathered using any single mode is incomplete. For example, when watching a movie, both the audio and the video are important for understanding the plot. Multimodal learning techniques are of particular interest for machine learning because processing data from multiple sources can be important to give the machine a more complete representation of the world.

Current machine learning methods for processing multimodal data often use deep neural networks. Although deep learning networks can provide great performance, they are often computationally intensive and require large amounts of training data. Additionally, the representations created when using neural networks are not often intuitive. In this thesis, we explore a novel and direct method that generates language context vectors from visual information without designing any new neural networks. Additionally, the language context that is generated consists of individual word scores making the output intuitive for a human to understand. Further, our method is

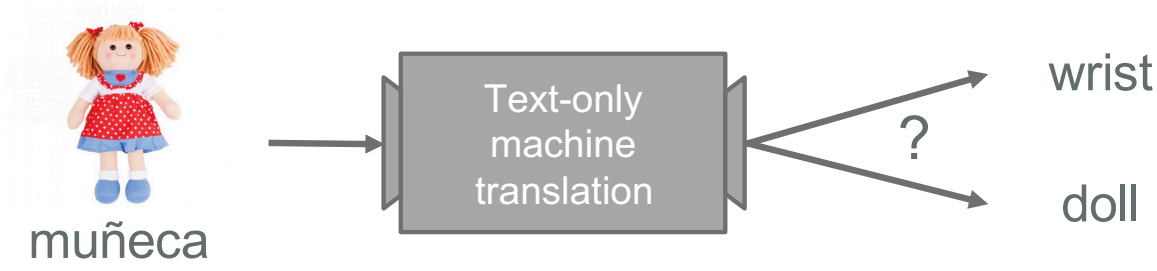


Figure 1.1: Translation is an example of where text context generated from visual features could be useful.

naturally language-invariant and can be used for any language with available training data.

The context vectors produced by our work could be used for a variety of applications. An example is to aid in machine translation tasks where visual context could provide insights into correct translations. One example is shown in Fig. 1.1 where the Spanish word "muñeca" means both wrist and doll. If translating to English, the correct translation depends on the context. If using an existing machine translation algorithm that provides a probability for each possible translation, our generated context vectors could be used to provide priors for the translations. This method would be much simpler and cheaper than having to retrain a new machine translation algorithm that uses visual features as input.

Another application could be in machine transcription where a computer converts audio signals of spoken words to text. Sometimes two different spoken words can sound very similar. An algorithm that has only an audio signal may not have high confidence for every spoken word. If visual information is available, our generated word score vectors could be used to break ties for these cases when the machine transcription algorithm cannot tell which word was spoken.

A third application could be to use the context vectors if searching for a video. In the case that a video does not have any preexisting textual information associated with it, the proposed method could be used to generate a list of terms likely to be associated with the video based on its visual features. These terms could then be used when searching a database of videos.

1.1 Contributions

Throughout this thesis, we explore a number of novel methods to represent and process multimodal data. Specifically, our contributions are:

- **A method to summarize a video using a weighted average of visual feature vectors based on scene detection.** This novel weighting method creates a summarization of the visual information of a video while reducing the impact of scene transitions that do not have valuable visual information.

- **A modification to the term frequency-inverse document frequency (TF–IDF) method for calculating word importance.** Traditional TF–IDF word scoring assumes that the words that appear in many documents are always less important. Our novel modification uses cluster information so that words must appear in many *types* of documents to be considered less important.

- **A method for using a nearest neighbors to predict what words are likely to be important for a video given its visual features.** This is the method used to predict context vectors for test videos under the assumption that videos that are nearby in the visual feature space are also nearby in the linguistic space.

- **A purity score that can be used to measure how well a test point fits the training space.** This score can be used as a confidence measure when predicting a context vector for a test video.

- **A method to visualize a 2-modal space.** This novel method uses a preexisting dimensionality reduction technique and a preexisting clustering technique to visualize how data points that are close in one modality are related to each other in a different modality.

1.2 Overview

To generate context vectors for videos, we start by generating a visual feature representation for each video. The visual feature representation is a vector that summarizes all of the useful visual information present in a video. Next, we generate a linguistic representation for the videos in the training set. The linguistic representation is the context vector that summarizes the useful information contained in a video transcript. To predict a context vector for a test video that does not have an associated transcript, a nearest neighbors approach is used. The visual feature of the test video is compared to the visual features of the training videos. The nearest neighbors are found, and their context vectors are averaged to create a predicted context for the test video. This predicted context vector consists of a score for every word in the dictionary and represents the predicted importance of each word for the test video.

The overall pipeline for generating a predicted context vector using visual features is shown in Fig. 1.2. In this figure, blue boxes with rounded corners represent algorithms that will be explored throughout this thesis. Each gray box represents a type of data that is used in the pipeline. For simplicity, only one variation of the pipeline

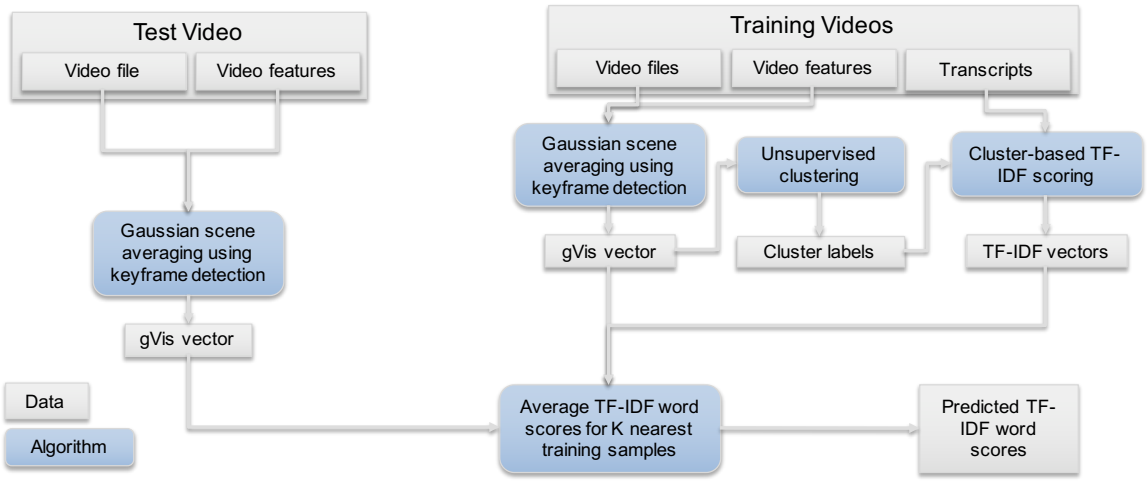


Figure 1.2: The full pipeline for predicting a word score vector from visual features.

is shown. Throughout this thesis we will explore 2 parts of the pipeline that can be varied each in 2 ways. This gives a total of 4 variations of the pipeline that we will discuss.

Chapter 2 discusses past work related to this thesis. In Chap. 3, we will introduce the How2 dataset. The How2 dataset is a multimodal dataset that will be used throughout this thesis. Then in Chap. 4, we will introduce the methods that are used to process the visual information of the videos. We show 2 methods for summarizing videos and a method for using unsupervised clustering to cluster videos based on their visual feature vectors. In Chap. 5, we discuss the processing that is done to make the provided text data usable. We also discuss term frequency–inverse document frequency (TF–IDF) scoring which is the method used generate the context vectors and introduce a modification to TF–IDF scoring that uses cluster information. Chapter 6 introduces the nearest neighbors averaging approach that is used to

predict word score vectors for test videos. In Chap. 7 we will further explore the relationship between visual information and linguistic information. For this exploration, we will also introduce a new representation for the linguistic information using a word embedding model. Chapter 8 shows the final predicted word scores for a number of example videos. Then Chap. 9 discusses methods that can be employed to reduce computational complexity of the pipeline. Finally, summarizations and conclusions are discussed in Chap. 10.

Chapter 2: Related Work

Previous work uses visual features in language processing. In [1], the authors use a deep learning approach to combine word and visual features to predict the next word in a sequence. To do this, they train recurrent neural networks on 64 million video segments and 1.2 billion tokens. They explore a variety of methods for combining visual and textual information within the neural network. The authors experimented with concatenation of feature vectors in the early, middle, and late stages of the network. They also experiment with combining feature vectors using a linear combination of feature vectors. Lastly, the tried to learn weights for combining feature vectors based on the current word. They found that training a neural network that combines visual and textual features in the middles of the network improves performance the most. The method presented in this thesis differs in two important ways. First, the language context is predicted using only visual features, and no other text or audio data. Second, our method does not require training a neural network which means our method can be used with significantly less training data.

The most similar previous work found comes from before deep learning was a popular, and traditional computer vision was used to relate words to visual information. In 2000, Mori et al. used traditional computer vision methods to extract visual features from images [7]. The visual features were clustered, and word histograms

were generated for each cluster, using nouns that came from the same documents as the images. The word histograms provided the probability of a certain word occurring, given that the image was part of a cluster. This thesis differs from the approach described because we use visual feature vectors generated by a pretrained neural network, and our visual information comes from videos rather than images. Additionally, we use term frequency-inverse document frequency word scores rather than histograms. Lastly, rather than dividing videos into discrete clusters, we use nearest neighbors to predict our word score vectors.

Previous work has been done to improve term frequency–inverse document frequency (TF–IDF) scores using cluster information. In [20], the authors use cluster information of documents to define a confidence and support value for each word. They then combine the confidence and support values to find feature words for a document. The modification to TF–IDF proposed in this thesis is a simpler approach that also uses cluster information. Furthermore, the approach used in this thesis can be implemented using an existing TF–IDF code package.

In this thesis we will explore a scene change detection algorithm that uses color histograms to compare frames. Previous work has also been done to find scene changes in a video. One such example is in FFmpeg [16], an open-source software for video and audio processing. The scene detection filter used in FFmpeg uses a sum of absolute differences (SAD) method to calculate when a scene has changed. As explored in Chap. 4, the SAD method for determining scene changes can be quite noisy. Due to this problem, we propose a new method that will use global color information to compare images. Using global color information allows the video to have large amounts of motion without interfering with the scene change detection.

When compared to similar work, this thesis introduces novel methods to expand on existing ideas. The first advantage of our method for producing context vectors is that it does not require a large amount of training data. Second, we use nearest neighbors rather than clustering to predict a unique context vector for each video. Finally, we expand on the existing TF–IDF word scoring method and introduce a new way to summarize the visual information of videos. These proposed improvements allow us to generate a context vector that gives textual information about a video given only its visual features.

Chapter 3: How2 Dataset

This thesis uses data from the How2 Dataset [13]. The How2 Dataset is a multimodal dataset created from instructional YouTube videos. Each video in the How2 dataset has a unique 11-character identifier that comes from its original YouTube identifier. The full How2 dataset contains approximately 2000 hours of video data from 79,114 videos. Not all of the videos in the full dataset have complete data available. For this thesis, we only use the 300-hour subset of the How2 dataset which contains 13,439 publicly available videos with complete multimodal data available.

According to the How2 paper [13], the 300-hour subset is divided into 13,168 training videos, 150 validation videos, and 175 test videos. The test videos are referred to as the "dev5" set in the downloads for the How2 data. Additionally, there are 169 videos in the dataset which were reserved for future use. The 169 held videos are not used in this thesis since they are not publicly available. For each of the videos in the 300-hour subset, the following data is available:

- Video files
- Cluster labels
- English video summaries/descriptions
- English word aligned ground-truth subtitles

- Portuguese translations of subtitles

The video files in the How2 dataset are multimodal since they contain both visual and audio information. Additionally, How2 provides a number of feature vectors from the data listed above. The available features are:

- Audio features
- Video visual feature vectors
- Video action features

3.1 Video Files

The video files are the focus of the How2 dataset. The videos were obtained from YouTube "using a keyword based spider" [13]. The method for using the spider is described in [19]. The video files provided in the How2 dataset come in 3 different formats: .mp4, .mkv, and .webm. All of these formats can be read in Python using OpenCV [4]. For this thesis, we will use OpenCV to load data from individual video frames. The instructional videos in the How2 dataset are originally in English. This means words spoken in the video are in English. The method explored in this thesis does not use the audio information from the video files.

Figure 3.1 shows 3 screenshots from videos in the How2 dataset. The first video, shown in Fig. 3.1a, shows someone explaining the design of eyeglasses. This video comes from the training dataset. Fig. 3.1b shows a second example video coming from the validation set. This video shows someone teaching how to do highlights at home. The final example video is shown in Fig. 3.1c. This example video shows someone explaining hearing aid technology and how to choose the correct hearing aid.

Transcription
I bet you thought hair was the only thing that can have a flat top, but not true, on contraire, eyeglasses can have…

Portuguese Translation
Posto que você pensou que o cabelo era o único que pode ter um topo plano, mas não é verdade. Contrariamente, os óculos podem ter…

Description
The top of flat top glasses goes straight across. Learn about flat top frame designs for eyeglasses in this free fashion video from a graduate student.

Set: train **How2 Topic:** Handy Work **ID:** Dnvt3h32J20

(a) A video about eyeglasses fashion designs.

Transcription
Hi I'm Lauren, I'm going to be discussing how to add natural highlights at home. If you do not feel as though you would like to go to a salon…

Portuguese Translation
Oi, eu sou Lauren, vou discutir como adicionar destaques naturais em casa. Se você não sente como se quisesse ir a um salão…

Description
To add natural highlights at home, purchase a highlighting kit from the drug store, mix the lightening solution and gently paint small sections of hair with a hair color brush…

Set: val **How2 Topic:** Hair Care **ID:** Eh2AVkAQsxI

(b) A video about hair styling from home.

Transcription
Hearing aid technology has improved dramatically in the last few years. Ten, twenty years ago, or more than twenty years ago. It didn't take off right away, because initially…

Portuguese Translation
A tecnologia de aparelhos auditivos melhorou dramaticamente nos últimos anos. Dez, vinte anos atrás, ou mais de vinte anos atrás. Ele não decolou imediatamente, porque inicialmente…

Description
The last few years have seen dramatic improvement in hearing aid technology. Learn about advances in hearing aid technology from an audiologist in this free health video.

Set: test **How2 Topic:** Electronic Music **ID:** FZ_tTg3JQI4

(c) A video about choosing hearing aids.

Figure 3.1: Three example videos from the How2 dataset. Faces have been blurred here for privacy.

3.2 Cluster Labels

The How2 dataset provides a cluster label for each video, referred to as the video's "topic." According to How2, the clusters were generated using Latent Dirichlet Allocation (LDA) [13]. The statistically driven LDA approach used words from the English subtitles for clustering. It is important to note that since the clusters provided in the How2 dataset are generated using the English subtitles, the clusters do not necessarily reflect how the visual information of the videos is related. The total number of clusters generated using LDA was 22. Each of these clusters was hand-labeled into a topic. The topics determined in the How2 dataset are shown in Table 3.1. The table also shows the total number of videos from each topic that are in the 300-hour subset of the How2 dataset. This table includes counts from the train, val, and test set. As seen in the table, distribution is unbalanced among topics.

The cluster topics provided by How2 were manually inspected for this thesis to ensure that the topics were reasonable. From each topic, 4-8 videos were randomly selected and viewed. We found that many of the videos sampled did not fit well into their assigned topic. Some examples are shown in Fig. 3.2. The first example in Fig. 3.2a is a video about car repair. This video was mislabeled and put in a cluster labeled as "stitching" in the How2 dataset. The second example, shown in Fig. 3.2b, is a video about how to buy a car. The How2 dataset had this video under the "mountaineering" topic. The final example of a poorly labeled video is shown in Fig. 3.2c. This video is an instructional video about modeling and does not match well to its assigned topic of "skateboarding." Due to the mismatch between the topic labels and the content of videos, the topic labels and How2 clusters were not used for the work in this thesis.

Topic	Number of videos
Relationships	1403
Yoga	1394
Stitching	1122
Handy Work	949
Cooking	933
Painting	957
Make up	769
Ball Games	684
Computers	596
Guitars	551
Health	518
Exercise	492
Racing	470
Skateboarding	376
Hair Care	374
Drums	349
Boxing	339
Fantasy	304
Drinks	249
Mountaineering	248
Gardening	209
Electronic Music	207

Table 3.1: Total number of videos belonging to each topic for the 300-hour subset of the How2 dataset.

Transcription
In this clip we will be reinstalling the fuel pump back into the tank and getting the tank ready to be put back into the vehicle. You see the technician has put the pump back into the tank, and now he is slipping the lock ring back into place…

Portuguese Translation
Neste clipe estaremos reinstalando a bomba de combustível de volta no tanque, e colocando o tanque pronto para ser colocado de volta no veículo. Você vê o técnico? colocou a bomba de volta no tanque, e agora ele está deslizando o anel de trava de volta no lugar…

Description
Learn how to reinstall the fuel pump after repairing or replacing it on a car with expert automotive tips in this free online auto repair and car maintenance video clip.

Set: train **How2 Topic:** Stitching **ID:** FPPeOFBaETQ

(a) A video about car repair was put in the "stitching" topic.

Transcription
Okay, that's under moderate acceleration on the freeway. So check this out again, and make sure it's something you could live with. There's no [INAUDIBLE] in here, that's just with the camera inside the car…

Portuguese Translation
Tudo bem, está sob aceleração moderada na auto-estrada. Então, verifique isso novamente e verifique se é algo com o qual você poderia viver. Não há [INAUDÍVEL] aqui, é só com a câmera dentro do carro…

Description
Learn how to focus on the sounds and feelings of the new car while test driving it with expert car buying advice from an experienced new car salesman in this free online used car video clip.

Set: train **How2 Topic:** Mountaineering **ID:** 3lbjzu3OnU0

(b) A video about buying a car was put in the "mountaineering" topic.

Transcription
Okay, ladies it's so important that you understand how to combine the catwalk with the full turn. And as we move forward, overlapping our feet, I'm going to cross right, turn…

Portuguese Translation
Ok, senhoras, é tão importante que você entenda como combinar a passarela com a volta completa. E à medida que avançamos, sobrepondo os pés, vou cruzar a direita, virar…

Description
Full turns on the runway can be used in combination to make striking model struts. Walk and turn on the catwalk like a runway model in this free modeling video.

Set: train **How2 Topic:** Skateboarding **ID:** F770xDw027w

(c) A video about modeling was put in the "skateboarding" topic.

Figure 3.2: Examples of videos sampled from the How2 dataset that did not fit their assigned clusters.

3.3 English Summaries

The English summaries provided in the How2 dataset came from the YouTube video creators. When uploading a YouTube video, the creator writes a description that appears as text below the video. These descriptions included in the How2 dataset and also referred to as summaries. Since the videos are originally in English, the summaries are also in English. The descriptions for three example videos are shown in Fig. 3.1.

3.4 English Subtitles

The How2 dataset provides text of the subtitles for all of the spoken words in each video. The set of subtitles is also referred to as the video transcription. Although the transcriptions correspond to the words spoken in the video, the words in the transcriptions to not always match exactly the words in the video. Sometimes small words are added, omitted, or changed in the transcription in such a way that does not affect the meaning of the sentence, but does make it so that the transcription does not match exactly the spoken words. A common example is the transcription omitting filler words such as "um" and omitting repeated words if the speaker stutters.

To enable proper exploitation of multimodal data, the How2 dataset performs alignment to match the words in the transcript to the correct time in the video when the words were spoken. This way, visual information, audio information, and text information can all be aligned and used together. The English subtitles for the How2 dataset were retrieved through the YouTube platform. The authors of the How2 dataset performed Viterbi alignment to match the transcriptions to video segments

[13]. Truncated examples of the subtitles provided with the How2 dataset are shown for three videos in Fig. 3.1.

3.5 Portuguese Subtitles

To enable the How2 dataset to be used in machine translation applications and allow for another modality, the English subtitles were translated to Portuguese. The creators of the How2 dataset crowdsourced the translation task by paying workers in Brazil and Portugal. Translations of the English subtitles were first generated using a machine translation neural network, then workers would watch the video and fix the automatically generated translations [13].

Since the videos are originally in English, it does not make sense to align Portuguese words with English audio. Nevertheless, the Portuguese translations are of aligned English phrases. This means that although Portuguese words may not align exactly with the video, the meanings of the translated phrases should still be aligned with the video data.

Portuguese subtitles are shown for the 3 example videos in Fig. 3.1. The beginning of the Portuguese subtitles is shown for each video. These translations correspond to the same portion of the transcriptions that are shown in English for each video.

3.6 Features

There are feature vectors are available for download on the GitHub repository for the How2 dataset. There is one set of feature vectors are extracted from audio and two available sets from the visual information.

The creators of the How2 dataset extracted audio features from the raw video audio for each video using Kaldi, an open-source speech recognition toolkit [12]. The extracted audio features are 43-dimensional and were normalized using Cepstral Mean and Variance Normalization (CMVN) [13].

The first set of video features available for the How2 dataset are the full visual feature vectors for each video. These files are available as numpy arrays and consist of a 2048-dimensional feature vector for every 16 non-overlapping frames of each video. The feature vectors were extracted using a 3-dimensional convolutional neural network [13] [5]. On the download page for the How2 dataset, these vectors are referred to as "Summarization action features in numpy arrays."

The second set of video features available for the How2 dataset are the sentence-level features. These features are referred to as "Action features in numpy arrays for MT and ASR" on the download page for the How2 dataset. These features consist of a single 2048-dimensional vector for each of the phrases in each video. The single vector is obtained by averaging the visual features that correspond to the same time of the video as the sentence [13].

3.7 Selected Data

For this thesis, we use only a portion of the features that are available from the 300-hour subset of the How2 dataset. First, we use the full visual feature vectors along with the video data to generate our own visual feature representation for each video. This processing is further described in Chap. 4. Second, we use the English and Portuguese transcripts for generating a linguistic representation of reach video. The linguistic processing is further described in Chap. 5.

It is important to note that there is a small amount of data missing from the files provided on the How2 download page. For this thesis, it should be noted that the full visual feature vector files were not provided for a total of 49 videos that are a part of the 300-hour subset. Of the 49 videos that do not have complete visual feature information, 1 is from the training set, and 48 are from the testing set. Since these videos have incomplete data, they are removed from analysis for purposes of this thesis. For a specific list of video identifiers that were removed see Appendix B. After removing the videos with incomplete visual information, we have a total of 13,167 training videos, 150 validation videos, and 127 test videos.

The 300-hour subset of the How2 dataset is also missing portions of the transcriptions for some of the training videos. The methods proposed in this thesis assume that the transcript contains words that are important for understanding the video. This means that our methods work even if the training videos do not have complete transcriptions. The training videos simply need to have portion of the transcription that contains valuable linguistic information. Only one video in the training set (video ID Ec9Qqd5EtTM) was determined to have no valuable linguistic information. This determination was made by the fact that all of the words in the transcription were stop words which have no important information. Since this video has no useful linguistic information, it was removed from the dataset. Other videos with incomplete transcriptions remained in the dataset, and all testing and validation videos were left in the dataset. This gives us a total of 13,166 training videos, 150 validation videos, and 127 test videos that are used throughout the rest of this thesis.

Chapter 4: Visual Processing

The visual feature vectors provided by How2 consist of one vector for every 16 frames. Since videos sometimes have a different number of frames, they have a different number of feature vectors. To compare videos, we first create a video summarization video feature. After summarization vectors are calculated, we employ a parameter free unsupervised clustering method to group videos.

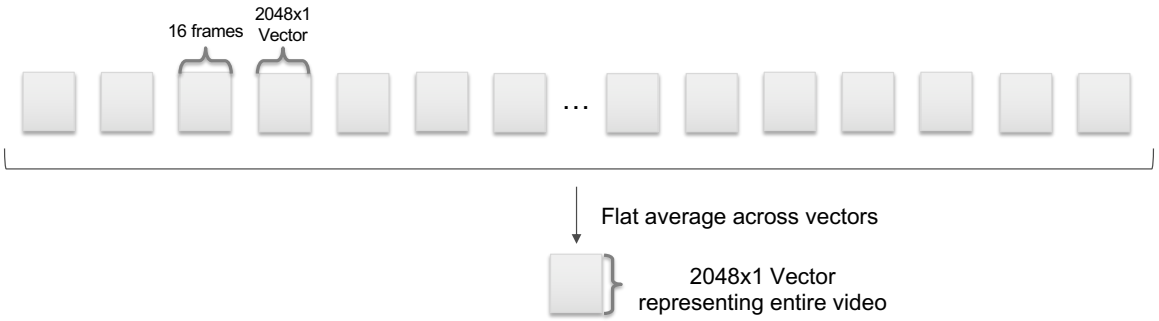
To summarize a video, we average its visual feature vectors across time. We introduce 2 methods of averaging: a flat average to create avgVis vectors, and a Gaussian scene weighted average to create gVis vectors.

4.1 avgVis Summarization

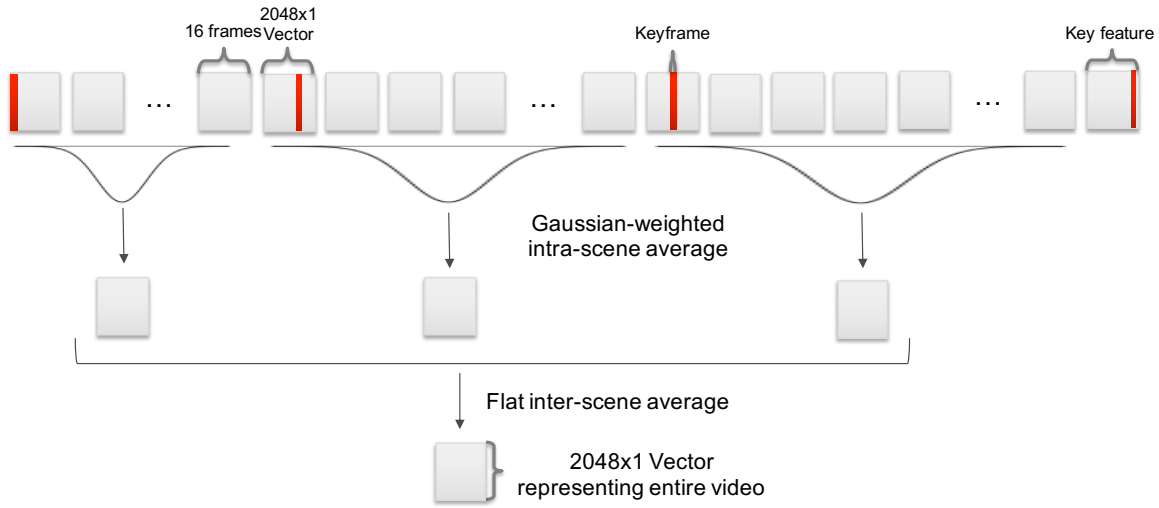
The first method is the trivial method that was used in the How2 paper. In this method, an unweighted average is taken across all of the visual feature vectors for the video. This method gives equal weight to all chunks of 16 frames, regardless of the content of those frames. Given all of the visual feature vectors of a video, $v_n \in \mathbb{R}^{2048}$ for $\{n \in \mathbb{Z} : 1 \leq n \leq N\}$ the avgVis vector is

$$v_{avg} = \frac{1}{N} \sum_{n=1}^{N} v_n \tag{4.1}$$

The avgVis vector, $v_{avg} \in \mathbb{R}^{2048}$, then represents a summary of all of the visual information in the video.

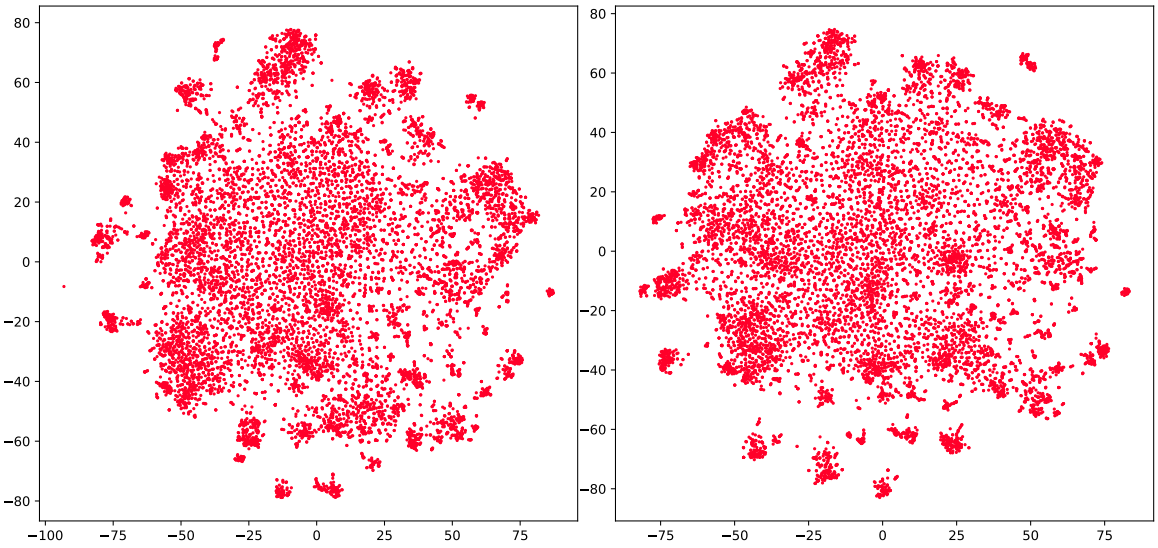


(a) A diagram showing how avgVis vectors are calculated.



(b) A diagram showing how gVis vectors are calculated.

Figure 4.1: A comparison of the avgVis and gVis visual feature averaging methods.



(a) Projections of the avgVis vectors.

(b) Projections of the gVis vectors.

Figure 4.2: A comparison of the avgVis and gVis visual feature vectors, projected from 2048-dimensions to 2 dimensions using t-distributed stochastic neighbor embedding (t-SNE) [17]. Each point in the plot represents a single video from the training data.

The avgVis method of visual feature summarization is very simple and not computationally intensive. To compute the avgVis vector for a video with N feature vectors, or equivalently $16 \times N$ frames, there is a total of only $2048 \times N$ additions and 2048 divisions. This makes the computation of avgVis vectors very fast and easy, especially compared to the gVis vectors that are discussed in the next section.

The avgVis method is shown in Fig. 4.1a. In this figure, each 2048-dimensional visual feature vector is represented by a gray rectangle. Since each visual feature vector is generated from 16 consecutive frames of the video, each gray rectangle also represents the visual information from 16 frames of the video. The unweighted averaging of the blocks across time is represented by the horizontal square bracket.

The final summarization vector is represented again by a gray box of equal size, since it is also a 2048-dimensional feature vector.

Projections of the avgVis vectors for the training data are shown in Fig. 4.2a. These projections were made using t-distributed Stochastic Neighbor Embedding (t-SNE) [17]. The details of the t-SNE algorithms along with the parameters that were used to generate these plots are further discussed in Chap. 7.

4.2 Scene Based gVis Summarization

The second method explored is a to average the scene vectors of the video. For this method, first visual feature vectors for each *scene* are calculated then they are averaged to produce a video summarization. To calculate visual feature vectors for each scene, scene change frames must be detected. These frames signify the start and end frames of a scene. To produce the visual feature vector for a scene, all of the visual feature vectors within the scene are averaged with a Gaussian weight. A Gaussian weighting is used because it is assumed that the most relevant information is in the middle of the scene, and the beginning and ending of the scene are scene transitions that contain less important visual information. We call this method gVis because the visual features for each scene are generated using Gaussian weighting. After each scene vector is calculated, the scene vectors are averaged with equal weights. This assumes that all scenes are equally important for the visual features of a video.

A specific method for calculating gVis vectors is as follows. First, a scene change detection algorithm is used to determine which frames of the video are scene changes. These frames are referred to as keyframes. Keyframes are frames directly after a scene change, or equivalently the first frame in the new scene. The first and last frames of

the video are also counted as keyframes. Since there are 16 frames per visual feature, the frame numbers of keyframes are converted to visual feature indices by dividing by 16 and rounding down (assuming the first frame number is 0). The visual features that contain at least one keyframe are referred to as "key features". It is important to note that although these feature vectors are referred to as "key features" it is assumed that these visual feature vectors actually contain very little valuable information. A scene is determined as the collection of visual features in between 2 consecutive key features. The key feature that begins the scene is included in the scene, but the key feature that contains the end of the scene is not included (see Fig. 4.1b). A Gaussian weighting with 3σ coverage is applied to each scene to give feature vectors in the middle of the scene higher weight than the features near the beginning and the end. Given the Gaussian function, or equivalently the probability density function of a zero-mean normal distribution with a standard deviation σ, as

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x}{\sigma}\right)^2} \tag{4.2}$$

and the visual feature vectors of the scene v_n for $\{n \in \mathbb{Z} : 1 \leq n \leq N\}$ the feature vector for a scene generated using Gaussian weights with coverage, c, would be:

$$v_s = \frac{\sum_{n=1}^{N} f(w_n) v_n}{\sum_{n=1}^{N} f(w_n)} \tag{4.3}$$

where w_n represent evenly spaced samples from $-c$ to c as given by

$$w_n = \frac{2c}{N-1}\left(n - \frac{N+1}{2}\right) \tag{4.4}$$

For a weighting with 3σ coverage, we set $c = 3\sigma$. The scene vector, v_s, is dependent only on the relationship between c and σ so the particular value of σ has no effect. This means we can set $\sigma = 1$ for simplicity and use the samples of the standard

Gaussian that are evenly spaced between $x = -3$ and $x = 3$. Using Python, w can be calculated by the command `w = np.linspace(-3, 3, N)`. After calculating the scene vectors v_s for $\{s \in \mathbb{Z} : 1 \leq s \leq M\}$ the gVis vector is

$$v_g = \frac{1}{M} \sum_{s=1}^{M} v_s \tag{4.5}$$

which is the unweighted average of the scene vectors.

The gVis method of feature summarization is shown in Fig. 4.1b. Like Fig. 4.1a, each gray rectangle represents a 2048-dimensional visual feature vector provided in the How2 dataset. In this figure, keyframes that signify the start of a new scene are shown in red. An example of a key feature is also labeled. The figure shows a total of 4 key features, and 3 scenes. The Gaussian curves below each scene represent the weighting of the different visual feature vectors described by 4.3. The horizontal square bracket represents the unweighted average of scene vectors described by 4.5.

Projections of the gVis vectors for the training data are shown in Fig. 4.2b. Like the avgVis projections, these projections were generated using t-SNE. The projections show that the visual space has a similar overall shape regardless of the summarization method. For both avgVis and gVis, the overall structure consists of one large blob in the center with smaller clusters around the edges. From this plot, there is not one summarization method that has a clear advantage. The details on t-SNE and the parameters that were used to generate these plots are further discussed in Chap. 7.

4.2.1 Scene Change Detection

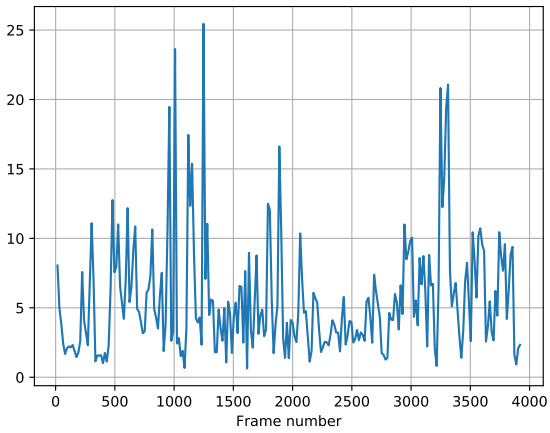
To determine the keyframes required for the gVis method of visual feature summarization, a method for determining scene changes is needed. Three methods for detecting scene changes were explored:

- Comparing visual feature vectors
- Comparing frames using sum of absolute differences
- Comparing frames using color histograms

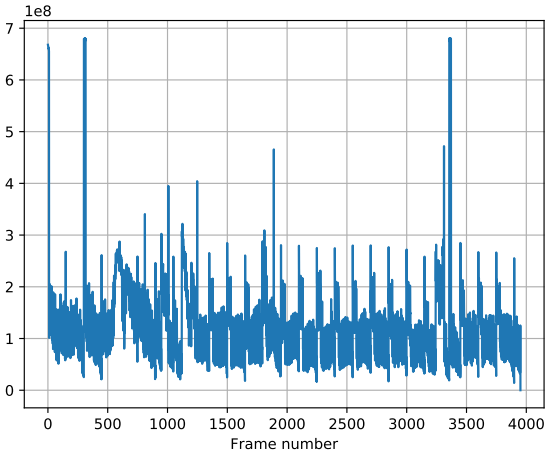
The first method was to compare the visual feature vectors for consecutive frame blocks directly. Since each 16-frame block has its own visual feature vector, this method was simple to implement by calculating the Euclidean distance between consecutive visual features. If the distance passed a threshold, then a scene change is detected. The visual feature comparison method was fast, since the vectors being compared were only length 2048. A disadvantage of using visual features to detect scene changes is that they can only detect changes on the order of 16 frames. The Euclidean distances between consecutive visual feature vectors for an example video are shown in Fig. 4.3a. In this figure, visual feature vector indices are converted to frame numbers by multiplying the indices by 16 since there are 16 frames for each visual feature vector. The figure shows that this method provides somewhat noisy scene change detections as there are no distinct sharp peaks.

The second method tested compared the sum of absolute differences (SAD) between each pixel in consecutive frames of the video. The is then compared to a threshold to determine if a scene change is detected. This method was slower than comparing feature vectors. This method was was also very sensitive to motion, which could easily get confused with a scene change. For 2 $N \times M$ frames X and Y, the sum of absolute differences would be

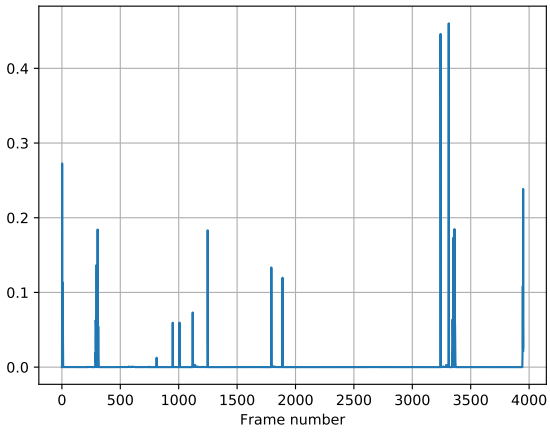
$$d = \sum_{n=1}^{N} \sum_{m=0}^{M} |X_{n,m} - Y_{n,m}| \tag{4.6}$$



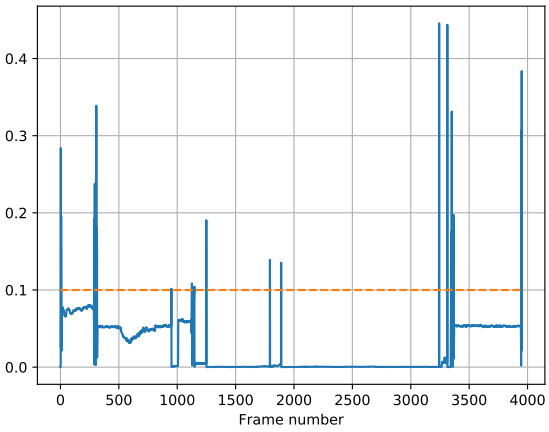
(a) Consecutive visual feature vector distances.



(b) Consecutive frame sum of absolute differences.



(c) Consecutive frame χ^2 distances from color histograms.



(d) χ^2 distance from reference frame using color histograms and a rolling threshold of 0.1.

Figure 4.3: A comparison of the different scene detection methods that for a sample video with I.D. bJUyq7y59ZA. The color histogram method gives the cleanest results.

The SAD between consecutive frames for an example video is shown in Fig. 4.3b. This figure shows that there are many peaks that come from motion rather than scene changes. This means SAD is a suboptimal method for scene change detection.

The third method compared the distribution of color data of consecutive frames. For this method, the color data for each frame needed to be converted to a histogram of color data. To isolate the brightness, a 3-D color histogram is created in the LAB color space, with 1 bin in the L channel, and 5 equally spaced bins for each of the A and B channels. This results in a total histogram of 25 bins for each frame of the video. The chi-squared distance is used to compare the distance between frames, since the chi-squared distance metric is bounded between 0 and 1 as shown by the equation

$$\chi^2 = \frac{1}{2}\sum_{b=1}^{N}\frac{(h_b - g_b)^2}{h_b + g_b} \tag{4.7}$$

where $N = 25$ is the number of bins, and g and h are L1-normalized color histograms from the 2 frames. Terms are omitted from the summation when the denominator $h_b + g_b = 0$. If the distance passes a threshold, then a scene change is detected. This color histogram method is the slowest but most reliable for detecting scene changes. To increase the speed of the algorithm, each frame is first scaled down to have its largest dimension be 100 pixels. The color histogram provided reliable results with sharp and tall peaks. Figure 4.3c shows the frame-to-frame χ^2 distances of color histograms. The tall peaks at scene changes show that a simple threshold can be used to reliably determine keyframes for calculating gVis vectors.

Since the color histogram method provided the most reliable results, it was used to detect scene changes for calculating gVis vectors. In order to make the algorithm more robust to slow transitions, a rolling scene change detection was implemented.

Using the rolling scene change detection, instead of comparing consecutive frames, each frame is compared to the frame of the previous scene detection. The frame of the previous scene detection is referred to as the reference frame. The rolling method allows for scene change detection, even for slow transitions. A threshold of 0.1 was used to detect scene changes. A visualization of the rolling threshold method is shown in Fig. 4.3d. The figure plots the χ^2 between each frame and its reference frame.

Generating the 3D color histogram for scene detection can take a long time. Even when the resolution of the video is downsampled to reduce computational complexity, there remains a high computational cost for reading and decoding each frame of each video. This limits the efficiency of computing the gVis vectors, as keyframe detection is an important step in calculating the visual feature vectors for individual scenes.

4.3 Clustering of Visual Features

The visual features were clustered using the FINCH hierarchical clustering method [14]. The FINCH algorithm requires no tuning of hyper-parameters and results in naturally sized clusters. The algorithm creates groups of points by looking at each point's nearest neighbor. Every point is connected to whichever other point it is closest to. For clusterable data, these connections create a set of separate clusters. This forms the first set of clusters in the hierarchy. The same process is repeated, but now instead of using the original points, the centers of each cluster are used. As the process repeats more, the number of clusters decreases, and the number of points in each cluster increases. For the visual features, Euclidean distance is used as the metric for calculating distances.

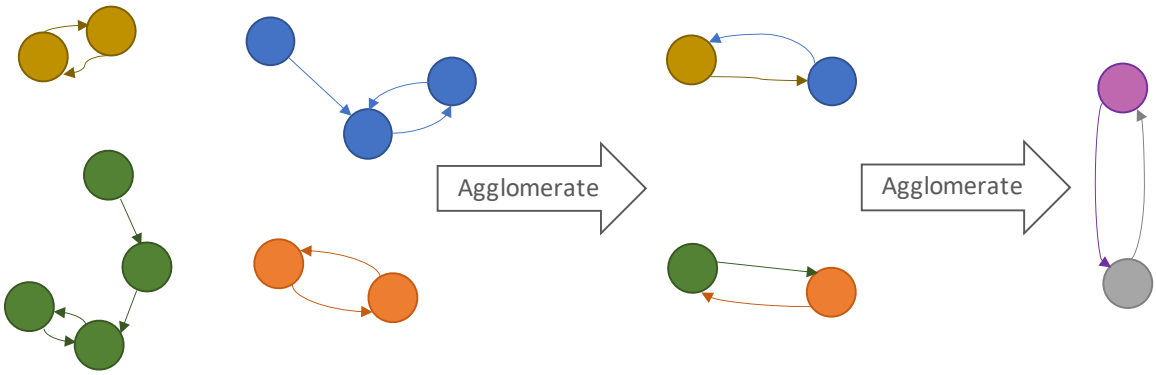


Figure 4.4: Example showing how the recursive Finch algorithm clusters points, then clusters clusters, based on nearest neighbor.

An example of the FINCH clustering process is shown in Fig. 4.4. At the first level shown in the example there are 11 data points. For clustering the visual features, each of these points would be in 2048-dimensional space, and each point would represent the visual information of the video. The arrows coming out of each point in the figure show the point's nearest neighbor. The 4 groups of points that share connections are the 4 clusters in the first level of the hierarchy. For cluster agglomeration, the centers of each of these clusters is used to give 4 points for the second level of the hierarchy. These points are clustered again to form 2 clusters. The process is repeated until all points end up in the same cluster.

Clustering the avgVis vectors of the training data resulted in a 5-level hierarchy. Each level of the hierarchy had the following number of clusters:

1. 3049 clusters for the first level
2. 572 clusters for the second level
3. 102 clusters for the third level

4. 23 clusters for the fourth level

5. 3 clusters for the fifth level

The 4th level of the hierarchy that contained 23 clusters is used for this thesis. This level of the hierarchy has the closest number of clusters to the original 22 clusters provided in the How2 dataset. Clustering the gVis vectors resulted in a slightly different 5-level hierarchy, as follows:

1. 2917 clusters for the first level

2. 549 clusters for the second level

3. 100 clusters for the third level

4. 13 clusters for the fourth level

5. 2 clusters for the fifth level

The 4th level of the gVis hierarchy which contained 13 clusters is the hierarchy level chosen for this thesis. This is the same level of as the avgVis vector and the level that contained closest number to the 22 provided How2 topics. Although we chose to use the FINCH clustering algorithm, other clustering algorithms could be used to cluster visual features

Chapter 5: Text Processing

For all the videos in the How2 dataset, the subtitles files are reformatted for usability. For training videos, punctuation and stop words are removed. Next, word stemming is used to remove suffixes from words to reduce the number of redundant words in the dictionary. After initial processing, word score vectors are calculated for training videos. Each training video is assigned a word score vector, $v \in \mathbb{R}^N$ where N is the total number of unique words in the training set (in other words, the dictionary length). The word score vector v, is calculated using term frequency–inverse document frequency (TF–IDF). For a particular video, the vector v gives a representation how important any particular word is for the video.

5.1 Preprocessing

The How2 dataset provides one transcription file per language per set. This gives a total of 6 files, since there are 2 languages (English and Portuguese) and 3 sets (training, validation, and testing). The transcription file contains all of the subtitles from all of the videos. To make the files easier to work with, the first step in the text processing is to separate these single large files into smaller files, one per video.

The process here is explained for the file provided "data/train/text.id.en" file, but the same procedure can be followed for the "dev5" and "val" datasets and for the

```
--7E2sU6zP4_8 And it has been wildly colored so you can look ...
--7E2sU6zP4_9 Here, actually, I have some samples of traced ...
--8pSDeC-fg_0 Hi.
--8pSDeC-fg_1 My name is Dr. Art Bowler.
--8pSDeC-fg_10 Are you always focused on the positive aspects ...
```

Figure 5.1: Example of 5 consecutive lines from the text.id.en file with English subtitles. Longer lines are truncated.

Portuguese translations which have file names that end in ".pt" rather than ".en". Each line contains an 11-character video identifier followed by an underscore then a phrase number then a space and a phrase as shown in Fig. 5.1. The order of the phrases in the file do not necessarily follow the same order the phrases are said in the video. The phrases from the same video are not necessarily on consecutive lines of the file. Each line of the text file is read into Python. The line is then split on the first space character to separate it into the identifier and the phrase. All of the phrases that come from the same video are appended to each other. Since each line ends in the newline character, no additional character is added between phrases. The result is written to a text file with the same name as the 11-character video identifier. Since the processes done in this thesis does not use order of words, the phrase number is ignored. An example of 5 lines from the file are shown in Fig. 5.1.

5.2 Text Cleanup

Before generating a word score vector for each training video, the text is cleaned. The first step in cleaning text is to remove punctuation. Most punctuation is converted to spaces. This is useful for when characters separate words, such as when

a dash is used. Additionally, it was observed that a space was not always put after punctuation, even if grammatically incorrect. For example in video bJ-sCKlDFYc, the phrase "to be called,this guy" appears without a space after the comma. An exception is made for punctuation that could be used as an apostrophe, which is simply removed without converting to a space. Removing apostrophes rather than converting them to spaces converts contractions to single words. These single-word contractions are also found in the dictionaries for word embedding models that will be further discussed in Chap. 7. The last exception of converting punctuation is for "%" and "$" these are left as is since these characters are more likely to provide useful information about the video. Numbers are also converted to spaces. See Table 5.1 for a details on how all the punctuation is modified. After all punctuation is removed, all the text in the transcript is converted to lower case.

After removing punctuation from the transcripts and converting them to lower case, stop words are removed from all of the transcripts. Stop words are common words, such as articles, that do not provide useful information. Common examples of stop words for English are the words "a", "and", and "the". The complete list of stop words (referred to as the stop words dictionary) comes from the `stop_words` package in Python [15]. For a list of all English and Portuguese stop words used, see Appendix D. Before removing stop words from the transcripts, we must have a dictionary of stop words that match the stop words in the transcripts. This means that our dictionary of stop words cannot contain any punctuation, since punctuation was already removed from the transcripts. The only punctuation in the stop words provided by the `stop_words` package is the apostrophe, so we remove the apostrophe from any stop words to convert them to the same single-word contractions that we

created in the transcripts. Once the punctuation is removed from the dictionary of stop words, we iterate through each word of each transcript. If the word is in the stop words dictionary, then it is removed from the transcript.

There are two primary reasons for removing punctuation from the transcripts before removing stop words from the transcripts. First, removing punctuation from the transcripts allows words that are adjacent to punctuation to still match a stop word. For example, if a transcript has the word "all." (with a period) and the stop word dictionary has the word "all" (without a period) then "all." (with a period) will not be removed from the transcript. If we remove the period before trying to remove stop words, then "all." (with a period) is converted to "all" (without a period) which is in the stop words dictionary. The second reason for removing punctuation first is that the video transcripts use a number of different characters to act as an apostrophe. For example, the contraction "can't" in the video transcript may use the character with Unicode I.D. 0x02BC (modifier letter apostrophe). If the stop word dictionary only has "can't" using Unicode I.D. 0x027 (apostrophe) then the word "can't" using Unicode I.D. 0x02BC (modifier letter apostrophe) will not be removed from the transcript. If instead we remove punctuation from both the stop words and the transcripts first, it does not matter what character was used as the apostrophe since the character is completely removed (i.e., the word in the transcript becomes "cant" and the word in the stop words dictionary becomes "cant").

The final cleanup stage after removing punctuation and stop words is word stemming. We use the Python package snowballstemmer [11]. Word stemming removes the ending of some words so that different related versions of a word are all mapped

to the same word. An example is given on the snowball stemmer website: "the English stemmer maps connection, connections, connective, connected, and connecting to connect" [11]. It is important to keep in mind that a stemmed word is not always a word itself, for example the word "picture" is stemmed to "pictur".

Table 5.1: A table showing what different punctuation is converted to.

Punctuation	**Unicode ID**	**Conversion**
!	U+0021	Space
”	U+0022	Space
#	U+0023	Space
$	U+0024	No change
%	U+0025	No change
&	U+0026	Space
’	U+0027	Removed
(	U+0028	Space
)	U+0029	Space
*	U+002A	Space
+	U+002B	Space
,	U+002C	Space
-	U+002D	Space
.	U+002E	Space
/	U+002F	Space
0	U+0030	Space
1	U+0031	Space
2	U+0032	Space
3	U+0033	Space
4	U+0034	Space
5	U+0035	Space
6	U+0036	Space
7	U+0037	Space
8	U+0038	Space
9	U+0039	Space
:	U+003A	Space
;	U+003B	Space
¡	U+003C	Space
=	U+003D	Space
¿	U+003E	Space

Continued on next page

Table 5.1 – continued from previous page

Punctuation	Unicode ID	Conversion
?	U+003F	Space
@	U+0040	Space
[	U+005B	Space
“	U+005C	Space
]	U+005D	Space
^	U+005E	Space
·	U+005F	Space
‘	U+0060	Removed
–	U+007B	Space
—	U+007C	Space
”	U+007D	Space
~	U+007E	Space
Apostrophe	U+02BC	Removed
En-dash	U+2013	Space
Em-dash	U+2014	Space
Left single quotation mark	U+2018	Space
Right single quotation mark	U+2019	Removed
Left double quotation mark	U+201C	Space
Right double quotation mark	U+201D	Space
Horizontal ellipsis	U+2026	Space

5.3 Word Score Vectors

After all preprocessing and cleanup, word score vectors are calculated for training videos. The word score vectors, also referred to as context vectors, are a representation of the linguistic information contained in the transcripts of the videos. Term frequency–inverse document frequency (TF–IDF) scores are used to assign word importance for each word for each video. When these scores are computed for every word in the dictionary, they make the word score vector.

Term frequency is the number of times a word appears in a specific document. Given a word, t, and a document, d, the term frequency is represented by

$$\mathrm{tf}(t, d) = \sum_{s \in d} \mathbb{1}_{t=s} \tag{5.1}$$

where $\mathbb{1}$ is an indicator function

$$\mathbb{1}_{t=s} = \begin{cases} 0 & t \neq s \\ 1 & t = s \end{cases} \tag{5.2}$$

Inverse document frequency is related to the inverse of how many documents contain a given term. Given a word, t, and a set of N documents, D, the inverse document frequency is written as

$$\mathrm{idf}(t, D) = \log \frac{1 + N}{1 + |\{d \in D : t \in d\}|} \tag{5.3}$$

The TF–IDF score is the product of the term frequency and the inverse document frequency

$$\text{tf-idf}(t, d, D) = \mathrm{tf}(t, d) \times \mathrm{idf}(t, D) \tag{5.4}$$

The TF–IDF metric gives a lower score to a word if it appears in many different documents. If many videos are about the same topic, then they might have similar important words. This leads to a problem with TF–IDF because the TF–IDF metric would see that a term occurs in many videos, so it would give the term a low IDF value. For example, if there are many videos about swimming in the dataset, then the word "swim" may have a low IDF score so the TF–IDF metric would imply that the word "swim" is not meaningful. To combat this problem, a modification to the standard TF–IDF model is proposed so that IDF only uses documents that are outside the given document's cluster.

Given a term, t, a document, d, a cluster of related documents, $\{C : d \in C\}$, and the full set of documents, D, the proposed cluster–based TF–IDF can be written as

$$\text{tf-idf}(t, d, D, C) = \text{tf}(t, d) \times \text{idf}(t, D_c) \tag{5.5}$$

where $D_c = \{d \in D : d \notin C\}$. With the proposed modification to TF–IDF, if all of the videos that are related to swimming are in the same cluster, then the IDF term does not penalize the word "swim" for each video that is in the cluster. As $|\{D\}| \to \infty$ and $|\{C\}| \to 0$ the cluster–based TF–IDF becomes the traditional TF–IDF.

The TF–IDF scores are stored in a vector where each element corresponds to one word. Each training video has its own TF–IDF vector. This is shown in Fig. 5.2. Additionally, each calculated TF–IDF score vector, s is L1-normalized so that $\|s\|_1 = 1$.

The TF–IDF vectors are calculated for both English and Portuguese words using the same method. The vectors could be generated for any language for which transcripts are available using the same method. It is important to note that the length of the TF–IDF vectors depends on the number of unique words in the training set (after stop words and word stemming). Another important observation is that the TF–IDF vectors are sparse. This sparsity is due to the fact that many words do not appear in a given video, making the term frequency zero for that word.

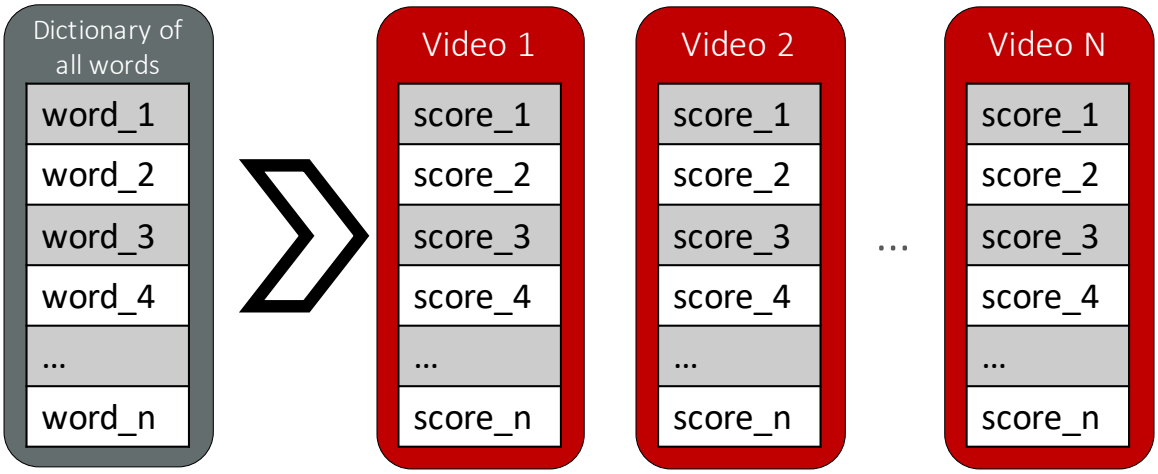


Figure 5.2: An illustration of the TF–IDF score vector.

Chapter 6: Inference

It is assumed that videos with similar visual features will have similar important words. Under this assumption, we use a nearest neighbors approach to predict a TF–IDF word score vector by averaging TF–IDF scores of neighboring training videos. Cross validation is used to determine the proper neighborhood size for prediction. We also introduce a purity score to give a confidence of how well a test video fits with the training data.

6.1 Nearest Neighbors Prediction

Nearest neighbors is a simple and robust approach to prediction. Given a test video, its visual feature vector is calculated as discussed in Chap. 4. The test video's visual feature, v_t is compared to the visual features of the training videos, v_i for $\{i \in \mathbb{Z} \ : 1 \leq i \leq N\}$ where N is the total number of training videos. This comparison is done using the Euclidean norm. The visual feature vectors, v_i, can be either avgVis vectors or gVis vectors. The K training videos with most similar visual feature vectors to the test video are selected. The KNN problem can be represented by the finding the set $\mathcal{K}$ such that

$$\arg\min_{\mathcal{K}} \sum_{i=1}^{N} \mathbb{1}_{i \in \mathcal{K}} \|v_i - v_t\|_2^2 \tag{6.1}$$

with the constraint that $|\{\mathcal{K}\}| = K$ and the elements of $\mathcal{K}$ are unique. The indicator function, $\mathbb{1}_{i \in \mathcal{K}}$, can be represented as

$$\mathbb{1}_{i \in \mathcal{K}} = \begin{cases} 0 & \text{if } i \notin \mathcal{K} \\ 1 & \text{if } i \in \mathcal{K} \end{cases} \tag{6.2}$$

The set $\mathcal{K}$ then contains the indices of the K nearest training videos in visual feature space. The TF–IDF vectors of the nearest training videos are averaged to create a predicted TF–IDF score vector for the test video as given by

$$\hat{s} = \frac{1}{K} \sum_{i \in \mathcal{K}} s_i \tag{6.3}$$

Where s_i corresponds to the word score vector for the i^{th} training video, which has a visual feature vector v_i. The word score vectors s_i can be computed using the proposed cluster-based TF–IDF scoring or the traditional TF–IDF scoring. When s_i is computed from cluster-based TF–IDF scoring, the clusters can be from either the avgVis or gVis vectors. The predicted score vector is L1-normalized for each word:

$$\hat{s_t} = \frac{\hat{s}}{\|\hat{s}\|_1} \tag{6.4}$$

6.2 Selecting K

The optimal value of K is selected using cross validation. We used the 150 validation videos provided by the How2 dataset. To find the optimal value of K we first need to calculate the true TF–IDF score vector, $s_t \in \mathbb{R}^D$, for each video in the validation set using the methods described in Chap. 5. When using the cluster-based TF–IDF approach, a cluster is assigned for the validation video based on the cluster of a mode of its neighbors. The assigned cluster C is given by the optimization problem

$$\arg\max_{C} \frac{1}{K} \sum_{i \in \mathcal{K}} \mathbb{1}_{v_i \in C} \tag{6.5}$$

When calculating the IDF values for the true word score vectors, only training videos are used.

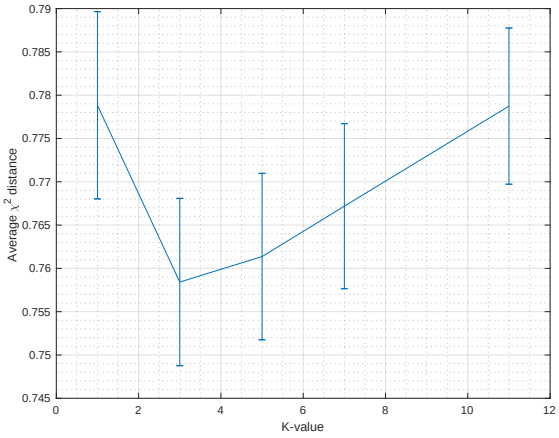
To select the optimal value of K we minimize the average χ^2 distance between the predicted and the true word score vectors by solving the optimization problem

$$\arg\min_{K} \frac{1}{M} \sum_{j=1}^{M} \chi^2(s_j, \hat{s_j}) \tag{6.6}$$

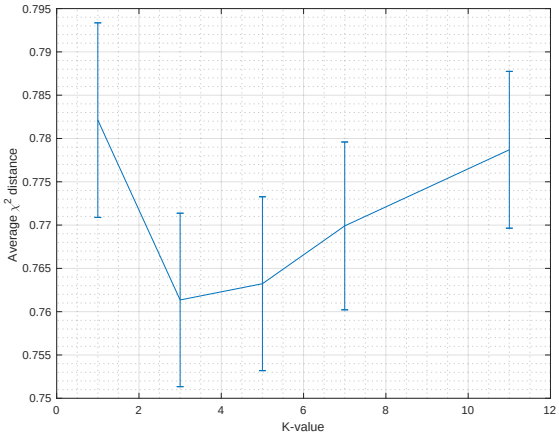
where

$$\chi^2(s_j, \hat{s_j}) = \frac{1}{2} \sum_{b=1}^{D} \frac{(s_{jb} - \hat{s_{jb}})^2}{s_{jb} + \hat{s_{jb}}} \tag{6.7}$$

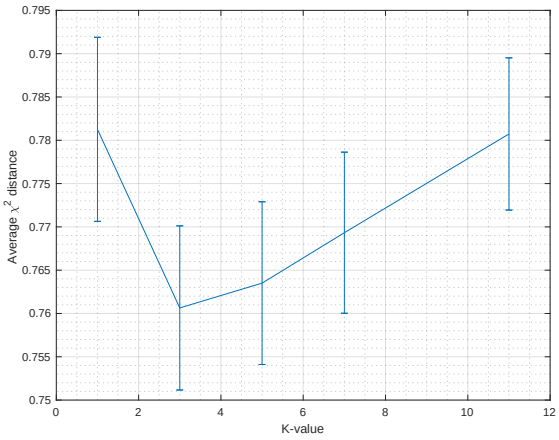
is the χ^2 distance between the predicted and the true normalized TF–IDF vectors of the j^{th} validation video, and there are M validation videos. We found that $K = 3$ is the optimal value for most combinations of visual feature averaging methods, TF–IDF scoring methods, and languages. The exception is when gVis vectors are used for inference on Portuguese TF–IDF vectors where $K = 5$ has a slightly lower average χ^2 distance. The average χ^2 distance between the predicted and true word score vectors for validation videos is shown in Fig. 6.1. The error bars in the figure show the variation of χ^2 distance among the validation videos. Since we have 2 methods to generate visual feature vectors for inference (avgVis and gVis) along with 2 methods to generate TF–IDF vectors (traditional and cluster-based) there are a total of 4 possible ways to predict word score vectors of a test video for a given language. We show cross validation for all 4 ways for each of the 2 languages in the dataset (English and Portuguese), hence the 8 subfigures in Fig. 6.1. We only test odd values of K so that we can assign a cluster to the validation video when computing the true cluster-based TF–IDF word scores.



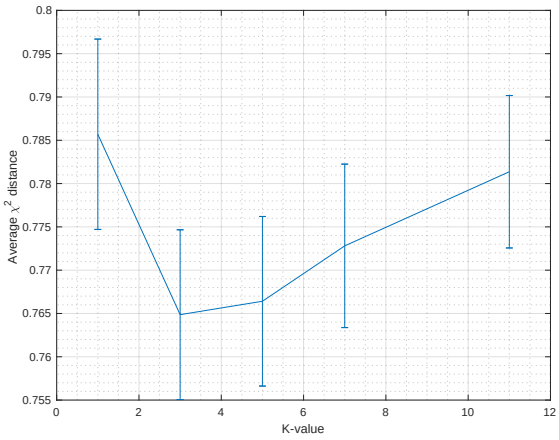
(a) English cluster-based TF–IDF vectors using FINCH avgVis clusters. The avgVis vectors are also used for inference.



(b) English cluster-based TF–IDF vectors using FINCH gVis clusters. The gVis vectors are also used for inference.

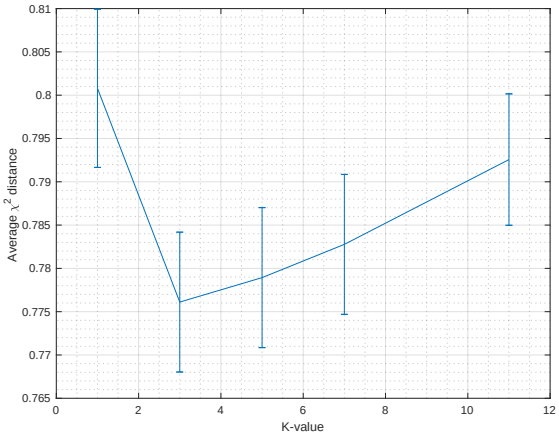


(c) English traditional TF–IDF vectors using avgVis vectors for inference.

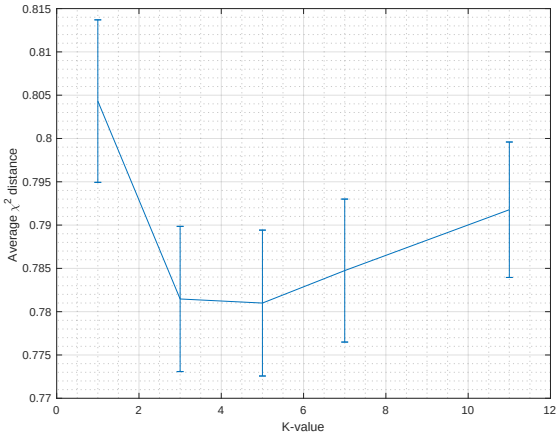


(d) English traditional TF–IDF vectors using gVis vectors for inference.

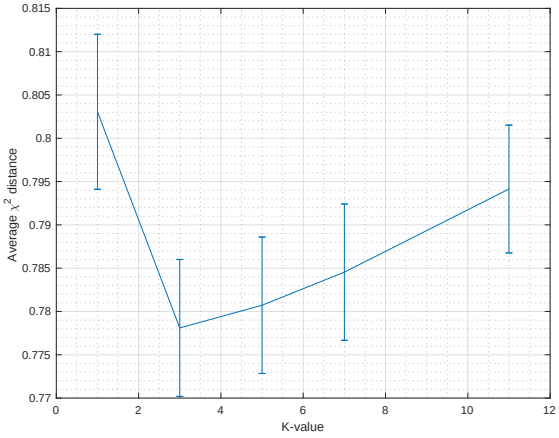
Figure 6.1: Analysis of cross validation to determine K for different methods.



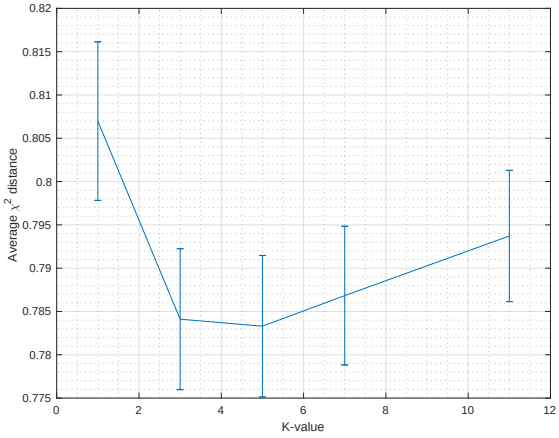
(e) Portuguese cluster-based TF–IDF vectors using FINCH avgVis clusters. The avgVis vectors are also used for inference.



(f) Portuguese cluster-based TF–IDF vectors using FINCH gVis clusters. The gVis vectors are also used for inference.



(g) Portuguese traditional TF–IDF vectors using avgVis vectors for inference.



(h) Portuguese traditional TF–IDF vectors using gVis vectors for inference.

Figure 6.1 (cont.): Analysis of cross validation to determine K for different methods.

One important note is that the value of K can have a slight effect on the true TF–IDF score vectors that are computed using the validation videos' transcripts when using cluster-based TF–IDF scores. This slight variation can come from the fact that changing the value of K could cause a validation video to be classified into a different cluster. When using cluster-based TF–IDF scoring, this change in cluster will affect the inverse documented frequencies, and therefore the word scores themselves. The overall effect of this phenomenon is negligible when compared to the effect that varying K has on the predicted word vectors.

6.3 Purity Score

In an effort to give a confidence of the prediction, we assign a purity score to each test video. Using the clusters generated by FINCH, the purity score gives a numerical value of how well the test video's visual feature, v_t, fits into a cluster. The purity score, p, is calculated by taking the maximum number of neighbors that share a cluster and dividing by the total number of neighbors, K. The purity score can be represented by

$$p = \max_{C} \frac{1}{K} \sum_{i \in \mathcal{K}} \mathbb{1}_{v_i \in C} \tag{6.8}$$

where C is a cluster generated by the FINCH algorithm. We choose C to be the cluster that has the most neighbors, and hence gives the highest value for the expression. We can see from Eqn. 6.8 that the purity score is bounded by the range $\frac{1}{K} \leq p \leq 1$. A purity score of 1 means all of the neighbors come from the same cluster so the test video's visual feature vector fits into the training space well. A purity score of $\frac{1}{K}$ means that every neighbor comes from a different cluster, so the test video's visual feature vector does not fit well into the training space. We expect videos with low

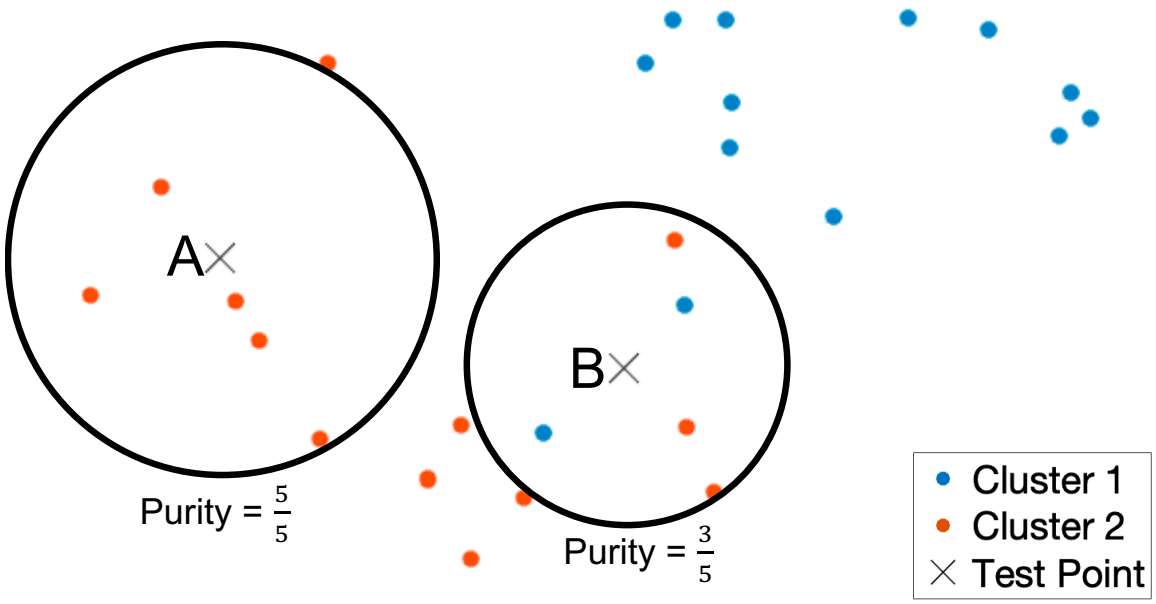
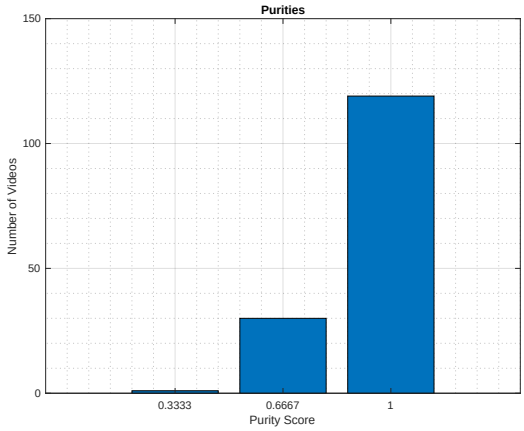


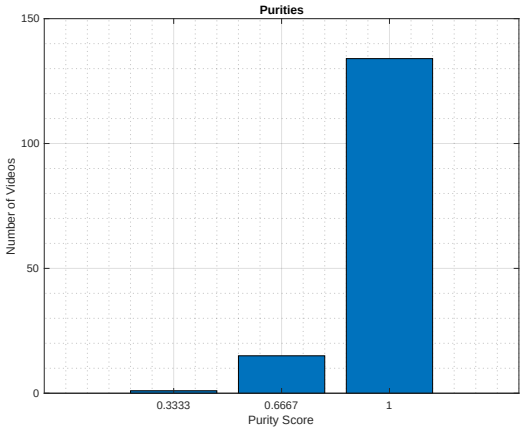
Figure 6.2: A 2D example of KNN with purity scores.

purity scores to have less reliable predicted word score vectors, $\hat{s}$. An 2D example of KNN with purity scores is shown in Fig. 6.2.

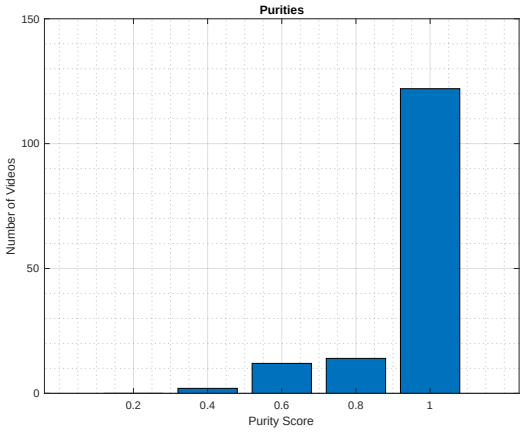
A plot showing the distribution of purity scores is shown in Fig. 6.3. The purity scores are shown for the validation videos. Since we can compute nearest neighbors for either the avgVis vectors or the gVis vectors, we show the purity scores for both cases. We show the purity score for each value of the optimal values K that were found. Since we found that K=3 is best for all cases when using avgVis vectors, we only show purities for avgVis vectors and K=3. For gVis vectors, we saw that when using English TF–IDF word score vectors K=3 was optimal, but when using Portuguese TF–IDF word score vectors K=5 was optimal. For this reason, purity scores for both K=3 and K=5 were included for the gVis vectors. We see that the purity score is 1



(a) The purity scores when using avgVis vectors and K=3.



(b) The purity scores when using gVis vectors and K=3.



(c) The purity scores when using gVis vectors and K=5.

Figure 6.3: The purity scores of videos in the validation set.

most of the time, regardless of the type of visual feature vector summarization used. This indicates that the nearest neighbors of the validation videos typically all come from the same cluster.

Chapter 7: Cluster Verification

In this chapter, we explore visualization techniques to explore the relationship between the space containing the visual information of training videos and the space containing the textual information of the training videos. We employ visualization techniques through dimensionality reduction techniques such as t-SNE to visualize how the two spaces relate to each other. We also apply a weighted average of word embeddings to create a textual representation vector different from TF–IDF score vectors, which are poorly suited for visualization as they are high dimensional and sparse.

7.1 Smooth Inverse Frequency

Although the visual feature vectors provide a mathematical representation for each video by their visual information, it was desirable to have a representation in the word space as well. Although TF–IDF score vectors are a mathematical representation of the linguistic space, they exist in a high dimensional space, and are sparse vectors. To develop a new representation, the method described for smooth inverse frequency (SIF) sentence embeddings was used [2]. This method was proposed to use a weighted average of word embedding vectors to form a sentence embedding vector. The same method is used to create video embedding vectors, referred to as SIF vectors.

Before generating SIF vectors, the text from the transcriptions is prepared. To prepare the text data, punctuation is removed, and all words are converted to lower case. This is the same method for text cleanup referred to in Chap. 5, except stop words are not removed and word stemming is not performed. This additional processing is avoided so that the embedding process has the complete word information from video.

The first step for creating SIF vectors is to have word embedding vectors. There are a number of available word embedding models trained on large datasets. These pretrained models often have a prebuilt dictionary available as well. For this project, we considered the dictionaries from 3 word embedding models:

- FastText [3]
- Global Vectors for Word Embedding (GloVe) [10]
- Paragram SL-999 (PSL) [18]

Each of these models has its own advantages and disadvantages. FastText learns from n-grams, or portions of words, to determine word meaning. This has the advantage that FastText can predict vectors for words that it has not seen before [3]. Since we use the prebuilt FastText dictionary in this thesis, we do not predict out-of-dictionary embeddings. We used the "crawl-300d-2M-subword" FastText dictionary which consists of "2 million word vectors trained with subword information on Common Crawl (600B tokens)" [3]. GloVe was trained by looking at how frequently certain words appear together within a document, so it has a global sense of how words work together [10]. We used the "glove.840B.300d" dictionary which was trained on Common Crawl of 840B tokens has a 2.2M word cased vocabulary and 300-dimensional vectors [10].

Embedding method	Number of missing words
FastText	1987
GloVe	1900
PSL	8406

Table 7.1: Number of missing words for different word embedding models.

The PSL model was trained using supervised data [18]. All of the word embedding models use a cosine similarity to determine similarity between words. When using unit vectors, the Euclidean distance can be used instead of cosine similarity. See Appendix A for a details.

To determine which word embedding methods best fit the How2 dataset, we count how many words from the training videos are missing from the model dictionary. The number of words that are missing from each of the different embedding models is shown in Table 7.1. The table shows that PSL had significantly more missing words than GloVe and FastText. We chose to use FastText vectors rather than GloVe vectors, because the provided FastText were unit vectors. This allows the embeddings to be compared using Euclidean distance directly. Note that there are other dictionaries available for FastText and GloVe. We chose the "crawl-300d-2M-subword" and "glove.840B.300d" dictionaries because they were the largest available for FastText and GloVe respectively.

The FastText word embeddings consist of a 300-dimensional unit vector for each word, $v_w \in \mathbb{R}^{300}$ and $\|v_w\|^2 = 1$. We wish to calculate a word embedding vector for a set of videos, S, using the SIF algorithm. First, for each video $\{s : s \in S\}$ the

weighted average of word embeddings is calculated as

$$v_s = \frac{1}{\{|w : w \in s|\}} \sum_{w \in s} \frac{a}{a + p(w)} v_w \tag{7.1}$$

where a is a hyper-parameter, and $p(w)$ is the global word frequency for the word w. The global word frequencies are obtained from the training data. Second, the singular vector with the largest singular value is removed from all video embeddings, v_s. Let matrix $X \in \mathbb{R}^{N \times M}$ be the matrix whose columns are v_s for all $s \in S$. The singular value decomposition (SVD) of X would be

$$X = U \Sigma V^T \tag{7.2}$$

where $U \in \mathbb{C}^{N \times N}$ and $V \in \mathbb{C}^{M \times M}$ are unitary, and $\Sigma \in \mathbb{R}^{N \times M}$ is a rectangular diagonal matrix with non-negative real entries. Assume that the values of Σ are in decreasing order. Let $U_c \in \mathbb{C}^{N \times C}$ be defined as the matrix whose columns are the first C columns of U. The matrix $Y \in \mathbb{R}^{N \times M}$ is computed as

$$Y = X - U_c U_c^T X \tag{7.3}$$

In the original proposed SIF algorithm, $C = 1$ is fixed. The final SIF embedding vectors are then the columns of Y after L2 normalization. The final L2 normalization is done so that embedding vectors can be compared using either the cosine distance metric or the Euclidean distance.

We explored a number of variations on the SIF algorithm. We explore the effect of the parameters a and C. We also explore what happens when an additional L2 normalization is done between the calculation of the v_s vectors and SVD component removal. Lastly, we compare what happens when the factor in calculating v_s is the reciprocal of the number of words in the video versus the reciprocal of the number of words in the video that have valid word embeddings from the model.

Manual exploration of the different variations was done by looking at a sample of video embeddings, and their nearest neighbors. The variation that results in embeddings whose nearest neighbors are most similar should be used. Since there is no ground truth for which videos are most similar, the exploration done was subjective. We found that most variations result in reasonable embeddings. Further analysis showed that the vectors v_s are roughly equal magnitude, meaning that normalization before SVD component removal has little effect. Few videos had words that were missing from the FastText dictionary, so the overall effect of normalizing based on the number of words vs the number of words that had valid embeddings was not noticeable. Lastly, although different values of a and C seemed to also give reasonable results, we used the values suggested in the SIF paper of $a = 10^{-3}$ and $C = 1$ for simplicity. It is important to note that a reasonable value of a should be between the maximum and minimum word frequencies, $p(w)$.

7.2 Word and Visual Spaces

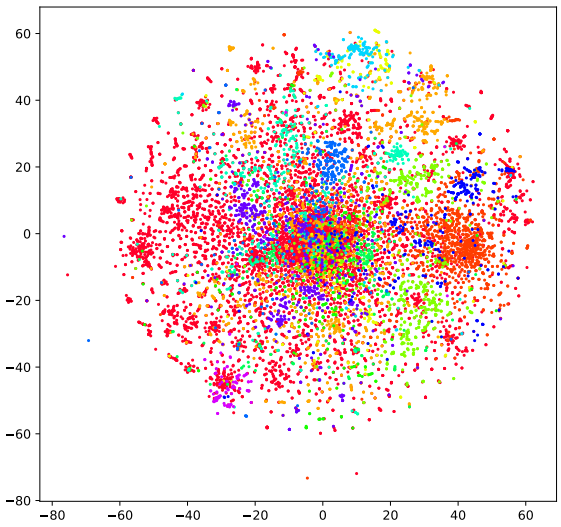
In order to visualize the relationship between the linguistic space and visual space, we use cluster labels generated in one space to color a t-SNE plot of projections in the other space. Cluster labels are generated using FINCH clustering. Figure 7.1 shows that for many videos, there appears to be a strong relationship between visual feature space and text space. This is shown particularly well when using the SIF vectors in Figures 7.1e to 7.1h when projected points are close and of the same color. We also see, however, that there is a large set of points with no strong relationship between visual feature space and text space. This is shown when projected points are close but have a large variation in color. This insightful visualization shows that although

we can predict useful text context using visual features for many videos, there are some videos where the visual information provides less useful information about the text that is likely to appear in the video.

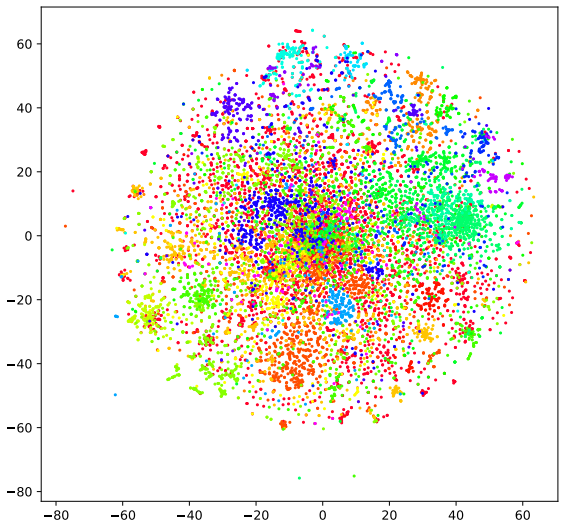
To reduce the number of possible combinations of word and feature representations, we only consider traditional English TF–IDF vectors for this experiment. Further experiments could be done using cluster-based TF–IDF vectors using clusters from either avgVis or gVis vectors. Experiments could also be done using the Portuguese TF–IDF vectors. We notice, however, that TF–IDF vectors are not easily clusterable nor do they provide very useful projections. Figures 7.1a and 7.1b show that the projections of the TF–IDF vectors are very uniform. This motivates the use of SIF vectors to represent the word space for visualization. Figures 7.1c and 7.1d show that there is one dominant cluster after clustering TF–IDF vectors. This makes the visualization less useful and further motivates the use of SIF vectors.

The t-SNE algorithm is a useful tool for reducing the dimensionality of data for visualization purposes. When generating projections, t-SNE attempts to plot similar feature vectors close to each other and dissimilar ones far from each other. This gives an overall idea of the shape of the data. For the examples shown, each point on the t-SNE plot represents an individual video.

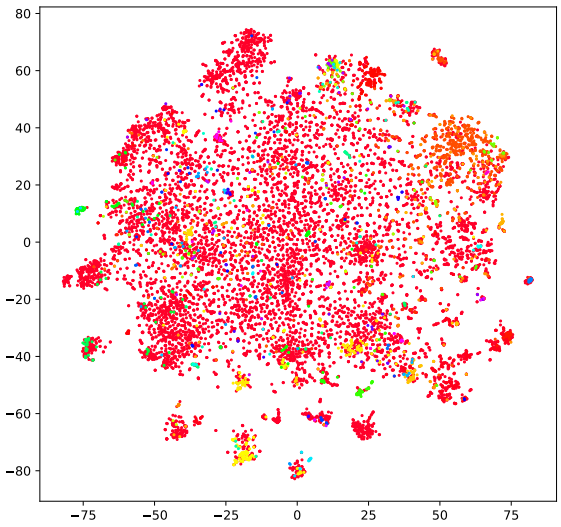
The t-SNE projections used for the plots generated in Fig. 7.1 were all generated with the hyper-parameters shown in Table 7.2. These parameters were chosen subjectively after comparing the plots for a number of hyper-parameters. The maximum number of iterations determines how long the algorithm can try to optimize the location of the points. The perplexity parameter is related to the number of nearest neighbors that are considered. The perplexity value can greatly affect the t-SNE



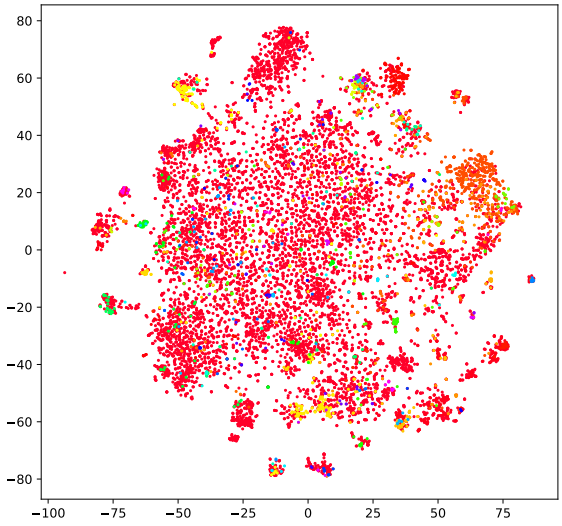
(a) t-SNE projections of TF–IDF vectors, with colors generated from gVis clusters.



(b) t-SNE projections of TF–IDF vectors, with colors generated from avgVis clusters.

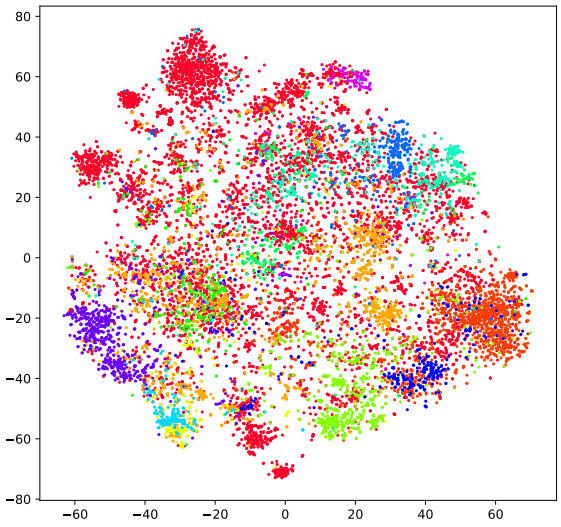


(c) t-SNE projections of gVis vectors, with colors generated from TF–IDF clusters.

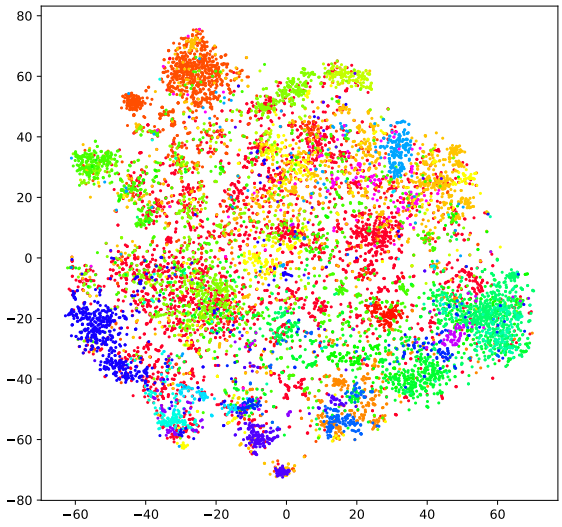


(d) t-SNE projections of avgVis vectors, with colors generated from TF–IDF clusters.

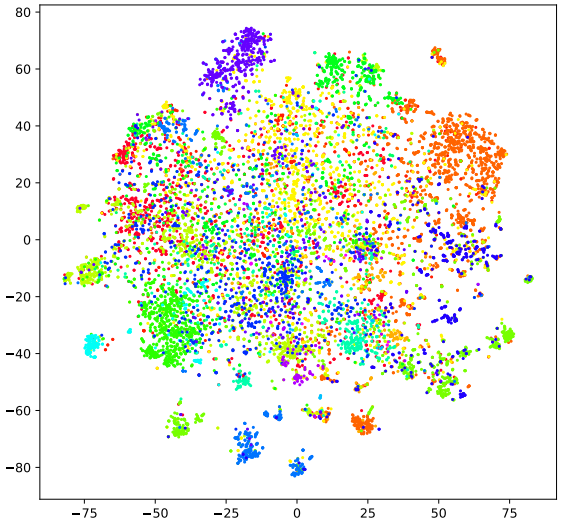
Figure 7.1: t-SNE projections of SIF vectors, with colors generated from avgVis clusters.



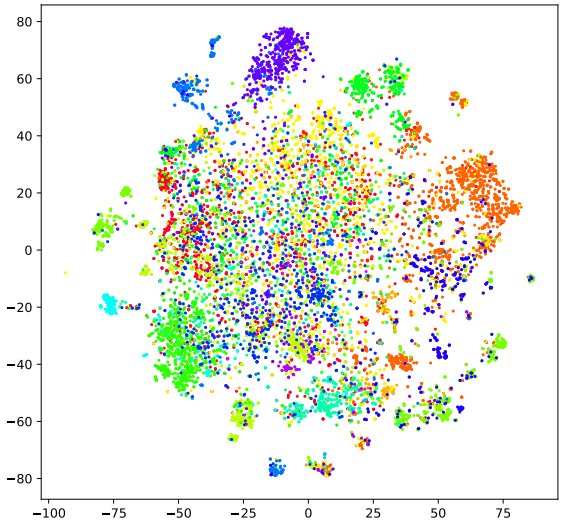
(e) t-SNE projections of SIF vectors, with colors generated from gVis clusters.



(f) t-SNE projections of SIF vectors, with colors generated from avgVis clusters.



(g) t-SNE projections of gVis vectors, with colors generated from SIF clusters.



(h) t-SNE projections of avgVis vectors, with colors generated from SIF clusters.

Figure 7.1 (cont.): A variety of t-SNE projections colored using clusters from a space different than the projected space.

Parameter	Value
Max Iterations	1000
Perplexity	50
Initialization	PCA
Learning Rate	200

Table 7.2: The selected hyper-parameters for t-SNE.

projections as a small value can create many tight clusters, even if no structure exists in the data. The initialization for the t-SNE algorithm was the points reduced by principal component analysis (PCA). Using PCA as initialization rather than a random initialization allows for reproducible projections that can be compared. The last selected hyper-parameter was the learning rate which affects how the t-SNE performs gradient decent to reach the optimal solution. Although we show t-SNE projections in this thesis, other dimensionality reduction tools could also be used. One such example is Uniform Manifold Approximation and Projection (UMAP) which attempts to model a manifold on which the data lies [6].

Chapter 8: Results

The final results for this thesis are the predicted TF–IDF vectors for the test data in the How2 dataset. The overall pipeline was shown earlier in Fig. 1.2. Variations of the pipeline are possible by changing on the type of visual feature vector used (avgVis or gVis) and the type of TF–IDF scoring (traditional or cluster-based). More variations could be explored, such as using clusters generated from gVis vectors to generate cluster-based TF–IDF scores but using avgVis vectors for calculating nearest neighbors, however we limit exploration to the 4 variations listed above. Additionally, when using cluster-based TF–IDF, the number of clusters can vary depending on what level of the FINCH heiarchy is used. For this thesis, we use the level of the FINCH hierarchy that is closest to 22 clusters, which is the number of original topics provided in the How2 dataset. This is the third level of the hierarchy for gVis (which has 13 clusters) and the third level of the hierarchy for avgVis (which has 23 clusters).

The pipeline is language invariant, which means word score vectors can be computed for any language that is available in the training data. For the How2 dataset, we have both English and Portuguese translations, so we compute word score vectors for both of these languages for every video. We consider the "predicted" word score vectors for a test video as the word scores generated using the nearest neighbors approach described in this thesis. We also compute "true" word score vectors for the

test video from the video's ground-truth transcript. The "true" word score vectors are used to validate our method and would not be available in practice since there would be no ground-truth transcript.

Considering all variations, we can produce a {`predicted, true`} word score vector in {`English, Portuguese`} using {`avgVis, gVis`} vectors and {`traditional, cluster-based`} TF–IDF scoring. This gives a total of 16 possible word clouds for each video. Note that when computing the true word scores using traditional TF–IDF scoring, visual feature vectors are unused, so the visual feature summarization method (avgVis/gVis) is irrelevant. This means there are a total of 14 unique word clouds per video.

We plot each of the 16 word score vectors as a word cloud using a Python package called `WordCloud` [8]. Each word cloud shows the words with the top 100 nonzero scores from the word score vector. The relative size of the words in the word cloud represents their relative word scores. The larger words have higher word scores, and the smaller words have lower word scores. This representation makes the word score vectors easy to visualize and provides an intuitive understanding of the language context generated. Although there are only 14 unique word score vectors, we display 16 word clouds for an easy side-by-side comparison of the "predicted" and "true" word score vectors for each of the 4 variations of the pipeline in each of the 2 languages. Since stemming is done before calculating word score vectors, the word clouds show word stems with punctuation removed.

We will show 3 example videos and their generated word clouds. All of the figures for these examples are shown at the end of this section. For some of the videos, the predicted word clouds may be better than expected due to bias in the How2

(a) A test video that is part of a video series.

(b) A training video from the same series as the test video.

(c) A second training video from the same series as the test video.

Figure 8.1: Example videos from a series in the How2 dataset.

dataset. It was observed that there are a large number of videos that are a part of a series. Often times these videos portray the same person in the same environment. If there is a training video that is part of the same video series as the test video, then the predicted word cloud could be informed by matching the person and the environment. The How2 dataset does not label which videos come from the same series, but manually viewing the nearest neighbors of a video can give some indication of if the video comes from a series. An example of this bias is shown in Fig. 8.1 where there were videos in the training set that are from the same series as the video in the test set. The series shown was about billiards.

One example test video is shown in Fig. 8.2 with its 16 word clouds. This figure shows an example video where a person is sitting and talking about diabetes. The video has a number of cuts to show a person eating, drinking, and sleeping. The predicted word clouds all show reasonable, but different, results with words with the highest word scores that relate to health and diabetes. The true word clouds have

high word scores for more specific words such as “vision”, “sugar”, and “blood”. The predicted word clouds give a reasonable context to describe what the video is about.

The Portuguese word clouds show similar performance. Some of the top words shown in the Portuguese word clouds are “sang”, “visã”, and “gordur”. The stem “sang” comes from the Portuguese word for “bleed”. The word “visã” means eyesight. “Gordur” is the stem of “gordura” which means fat. We see that the top Portuguese words correspond well to the top English words.

A second example test video is shown in Fig. 8.3. This figure shows an example video where a person is outside talking about proper hand ware for making a snowball. For this video we notice the word “board” has a high score when using avgVis vectors. Upon investigating the nearest neighbors of this video’s avgVis features, we found that there were videos about snowboarding. The snowboarding videos were visually similar to this test video as they are all outside and have a lot of snow. For this example video, using gVis vectors appears to give better results for the predicted word score vectors since the word “board” has a much lower score. Like in the first example, the word clouds shown for this video give a good context for the content in the video.

The Portuguese word clouds match well to the English word clouds for this second example video. We see that some of the top Portuguese words are “nev”, “bol”, “luv”, and “plac”. The stem “nev” comes from the Portuguese word for snow. The stem “bol” comes from the word for ball. “Luv” is the stem of the word that means glove. Lastly, “plac” comes from the word that means board.

A final example video from the test dataset is shown in Fig. 8.4. In this video, a person is practicing baseball. This vidoe’s predicted word clouds again provide

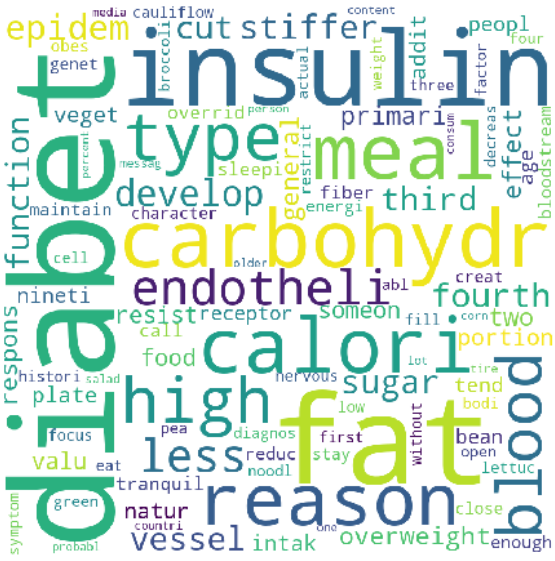
a reasonable context about what is going on in the video. We see that the words with the highest scores, such as “ball”, “pitch”, and “hit” are all related to baseball. For this video, it is difficult to say if the avgVis or the gVis vectors are a better representation for video context since both provide reasonable results.

Again, the Portuguese word clouds correspond well to the English ones. We see the words “beisbol”, “acert”, and “contat” which come from the English words for “baseball”, “hit”, and “contact” respectively. These translations are present with high scores in the English word clouds as well.

Overall, we find that any of the proposed methods for predicting word score vectors for a test video provide reasonable results. Different variations of the proposed pipeline all provide different, but reasonable, word score vectors. Since there are 3 unique ways to generate a true word score vector for a video given a language, it difficult to assess the best variation for predicting TF–IDF scores. Because there is no objective ground-truth for word score vectors, we can only provide a qualitative analysis to compare the different variations.

(a) A screenshot of the example video (video ID g0UUmEkoT1w). Face is blurred for privacy.

Figure 8.2: An example video and its 16 word score vectors, each represented by a word cloud.



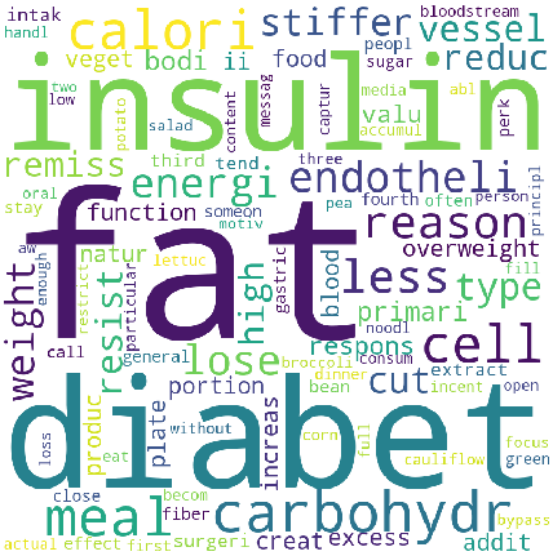
(b) Predicted English word score vector using avgVis vectors and traditional TF–IDF.

(c) True English word score vector using avgVis vectors and traditional TF–IDF

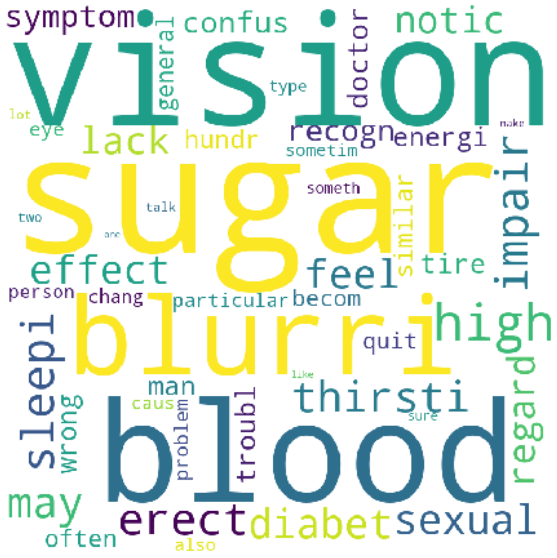
(d) Predicted Portuguese word score vector using avgVis vectors and traditional TF–IDF.

(e) True Portuguese word score vector using avgVis vectors and traditional TF–IDF.

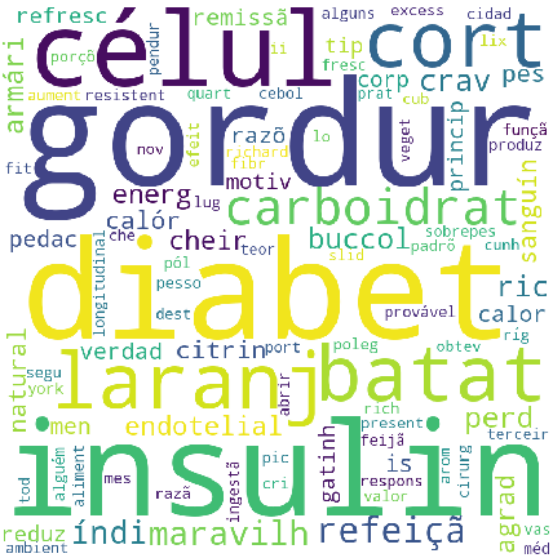
Figure 8.2 (cont.): An example video and its 16 word score vectors, each represented by a word cloud.



(f) Predicted English word score vector using gVis vectors and traditional TF–IDF.



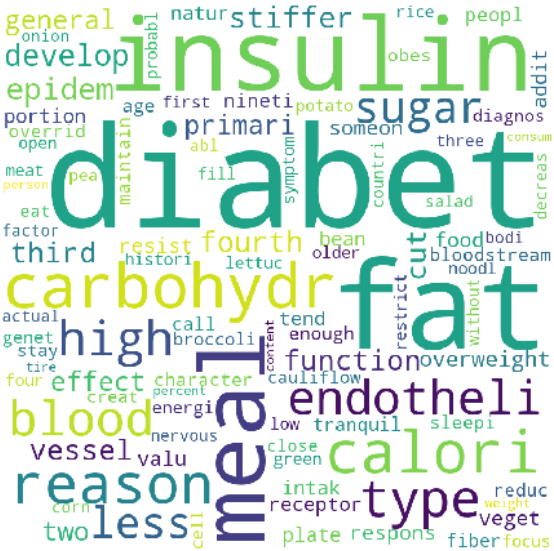
(g) True English word score vector using gVis vectors and traditional TF–IDF.



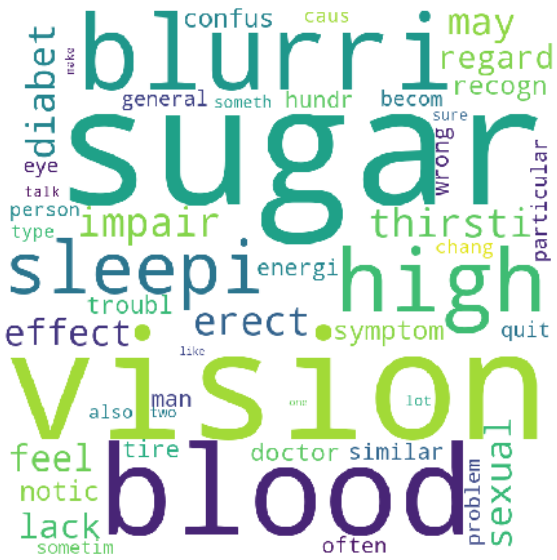
(h) Predicted Portuguese word score vector using gVis vectors and traditional TF–IDF.

(i) True Portuguese word score vector using gVis vectors and traditional TF–IDF.

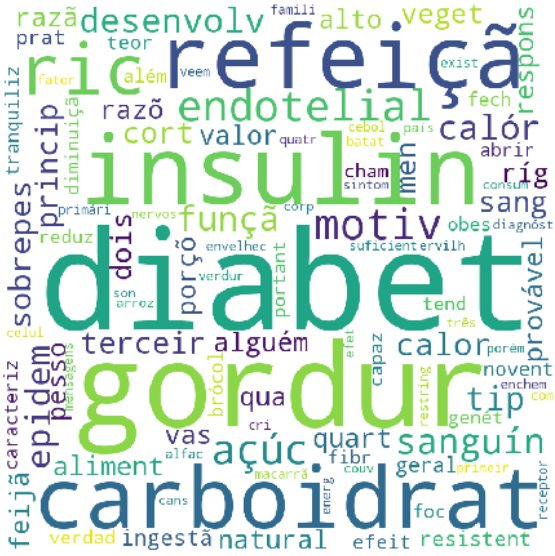
Figure 8.2 (cont.): An example video and its 16 word score vectors, each represented by a word cloud.



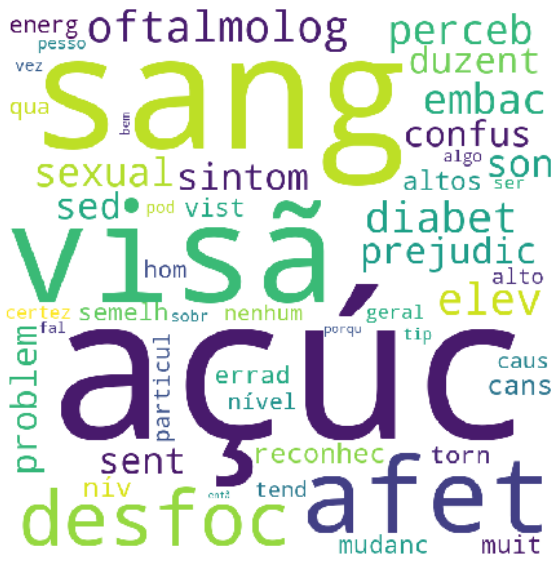
(j) Predicted English word score vector using avgVis vectors and cluster-based TF–IDF.



(k) True English word score vector using avgVis vectors and cluster-based TF–IDF.

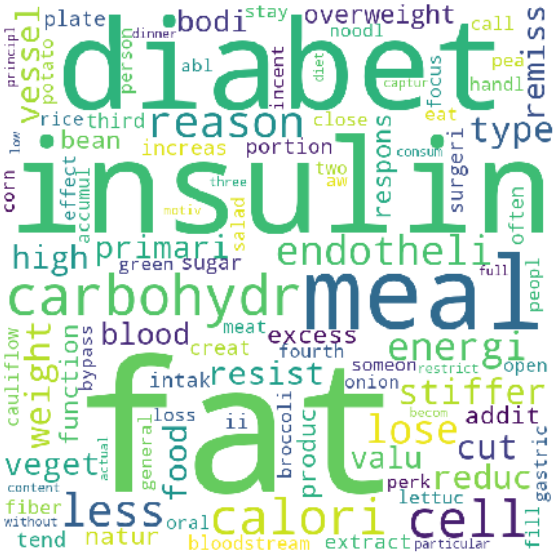


(l) Predicted Portuguese word score vector using avgVis vectors and cluster-based TF–IDF.



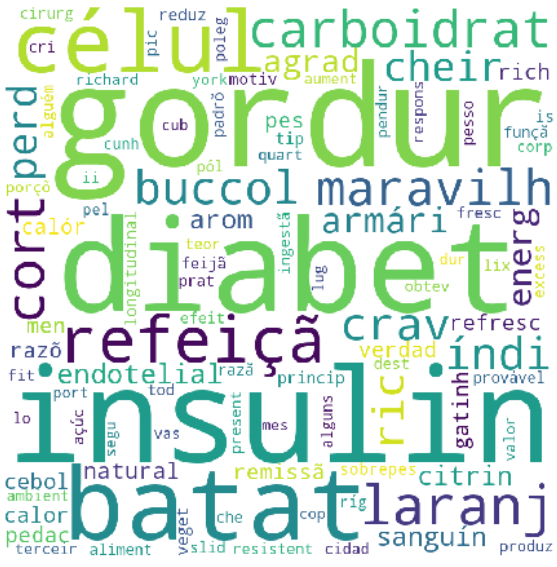
(m) True Portuguese word score vector using avgVis vectors and cluster-based TF–IDF.

Figure 8.2 (cont.): An example video and its 16 word score vectors, each represented by a word cloud.



(n) Predicted English word score vector using gVis vectors and cluster-based TF–IDF.

(o) True English word score vector using gVis vectors and cluster-based TF–IDF.



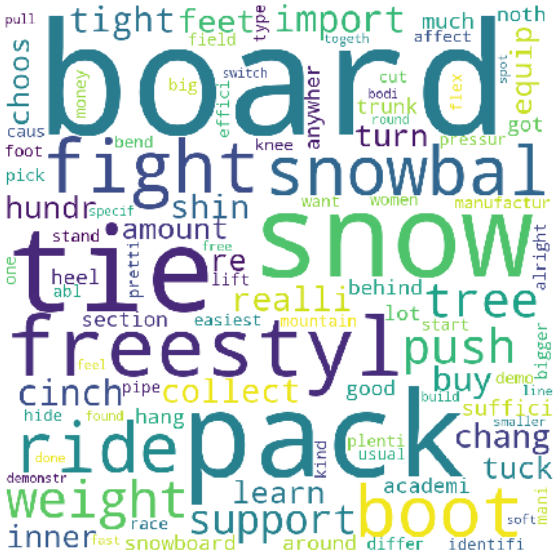
(p) Predicted Portuguese word score vector using gVis vectors and cluster-based TF–IDF.

(q) True Portuguese word score vector using gVis vectors and cluster-based TF–IDF.

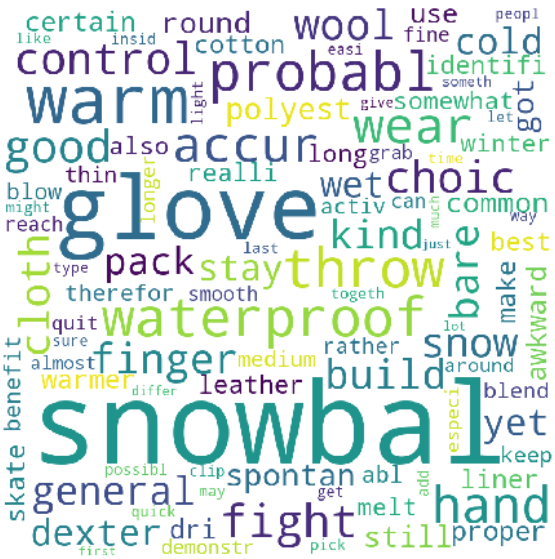
Figure 8.2 (cont.): An example video and its 16 word score vectors, each represented by a word cloud.

(a) A screenshot of the example video (video ID fzOH00UZg84). Face is blurred for privacy.

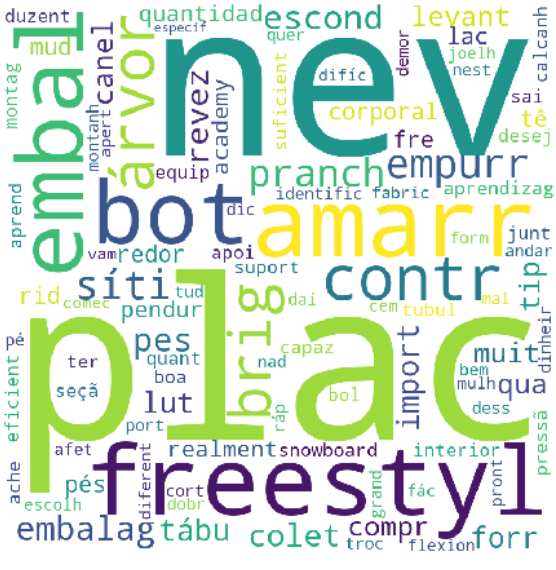
Figure 8.3: An example video and its 16 word score vectors, each represented by a word cloud.



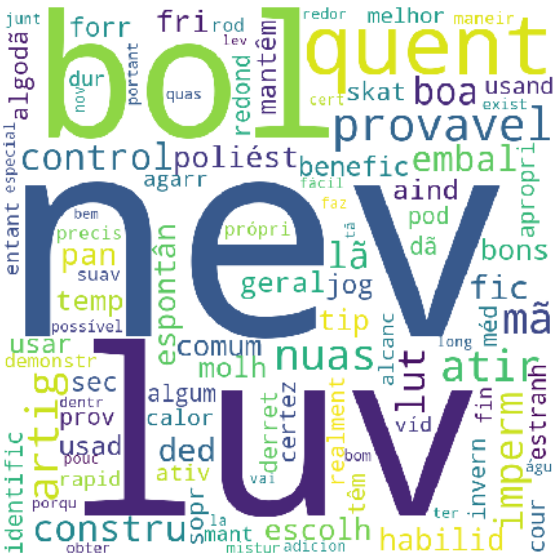
(b) Predicted English word score vector using avgVis vectors and traditional TF–IDF.



(c) True English word score vector using avgVis vectors and traditional TF–IDF

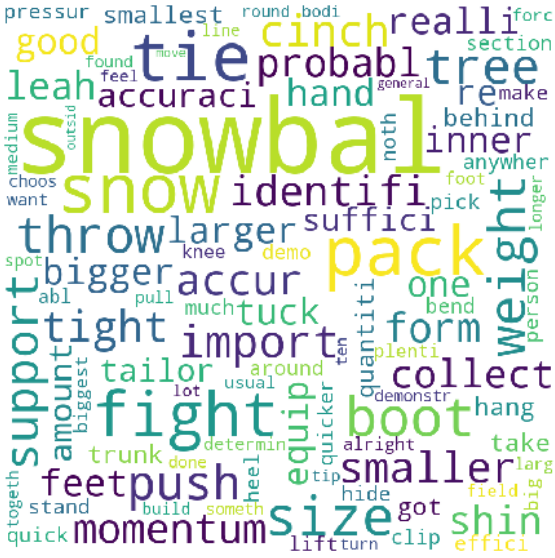


(d) Predicted Portuguese word score vector using avgVis vectors and traditional TF–IDF.



(e) True Portuguese word score vector using avgVis vectors and traditional TF–IDF.

Figure 8.3 (cont.): An example video and its 16 word score vectors, each represented by a word cloud.



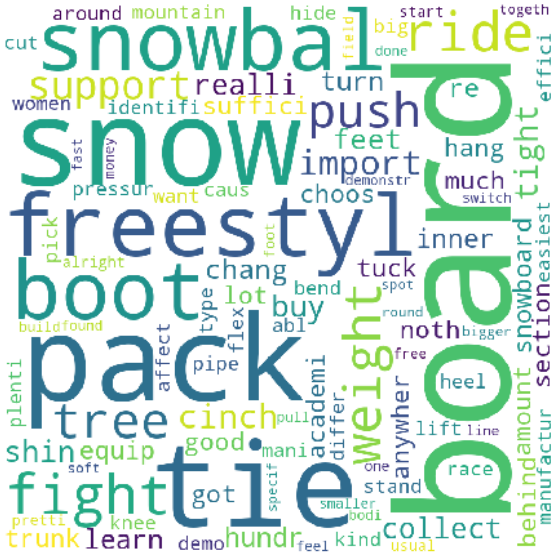
(f) Predicted English word score vector using gVis vectors and traditional TF–IDF.

(g) True English word score vector using gVis vectors and traditional TF–IDF.

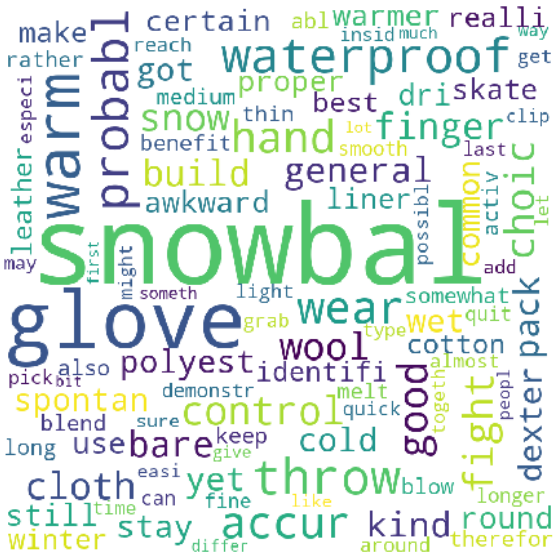
(h) Predicted Portuguese word score vector using gVis vectors and traditional TF–IDF.

(i) True Portuguese word score vector using gVis vectors and traditional TF–IDF.

Figure 8.3 (cont.): An example video and its 16 word score vectors, each represented by a word cloud.



(j) Predicted English word score vector using avgVis vectors and cluster-based TF–IDF.

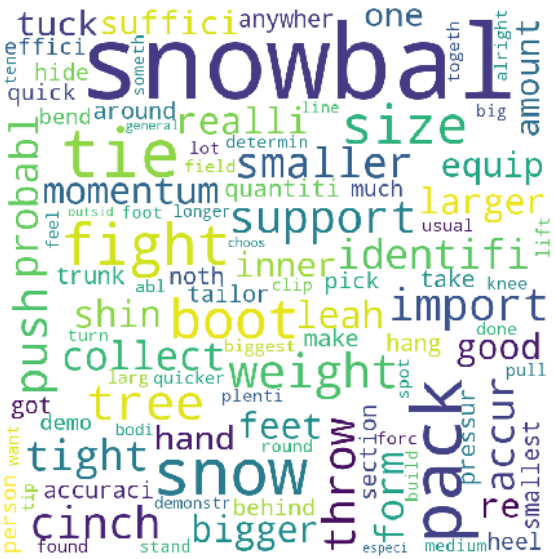


(k) True English word score vector using avgVis vectors and cluster-based TF–IDF.

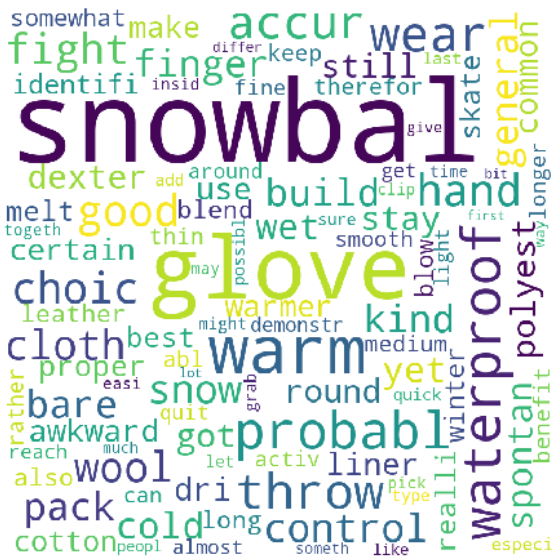
(l) Predicted Portuguese word score vector using avgVis vectors and cluster-based TF–IDF.

(m) True Portuguese word score vector using avgVis vectors and cluster-based TF–IDF.

Figure 8.3 (cont.): An example video and its 16 word score vectors, each represented by a word cloud.

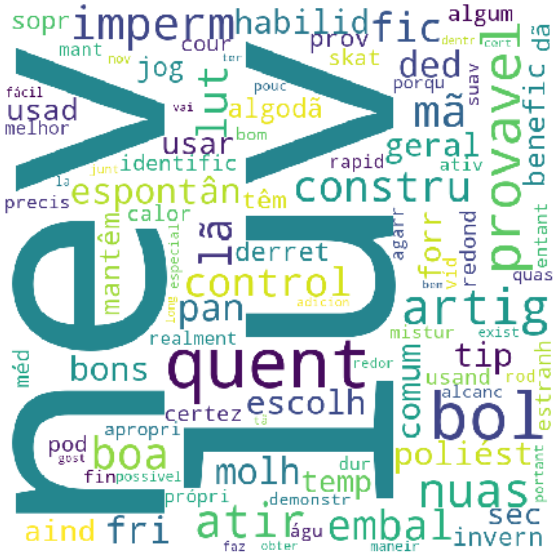


(n) Predicted English word score vector using gVis vectors and cluster-based TF–IDF.



(o) True English word score vector using gVis vectors and cluster-based TF–IDF.

(p) Predicted Portuguese word score vector using gVis vectors and cluster-based TF–IDF.



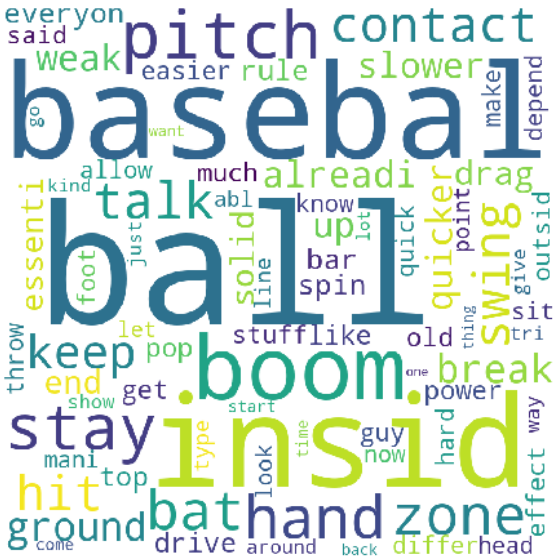
(q) True Portuguese word score vector using gVis vectors and cluster-based TF–IDF.

Figure 8.3 (cont.): An example video and its 16 word score vectors, each represented by a word cloud.

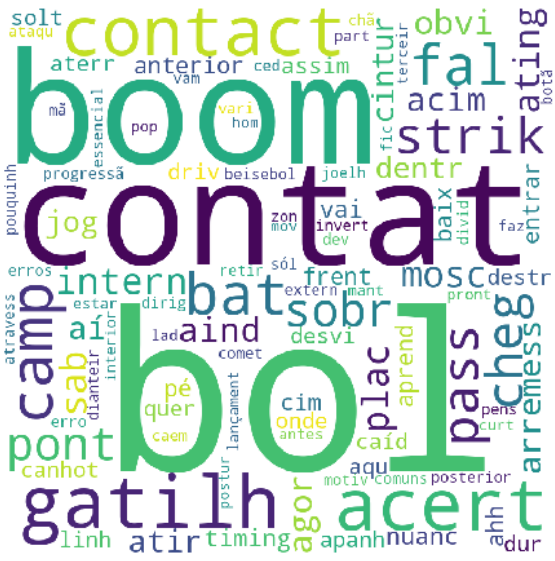
(a) A screenshot of the example video (video ID 92V3oH63zbQ). Face is blurred for privacy.

Figure 8.4: An example video and its 16 word score vectors, each represented by a word cloud.

(b) Predicted English word score vector using avgVis vectors and traditional TF–IDF.



(c) True English word score vector using avgVis vectors and traditional TF–IDF



(d) Predicted Portuguese word score vector using avgVis vectors and traditional TF–IDF.

(e) True Portuguese word score vector using avgVis vectors and traditional TF–IDF.

Figure 8.4 (cont.): An example video and its 16 word score vectors, each represented by a word cloud.

(f) Predicted English word score vector using gVis vectors and traditional TF–IDF.

(g) True English word score vector using gVis vectors and traditional TF–IDF.

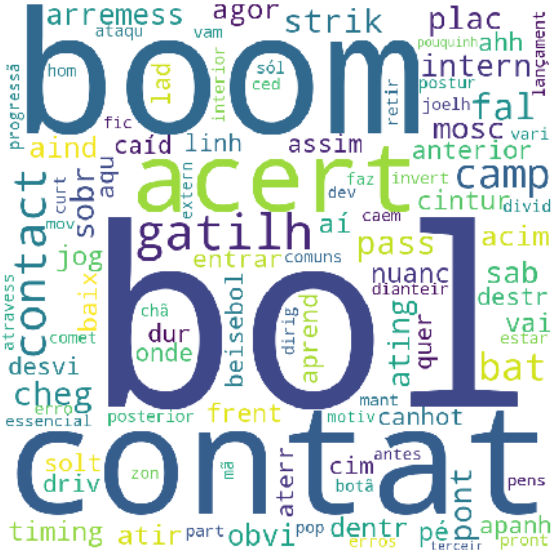
(h) Predicted Portuguese word score vector using gVis vectors and traditional TF–IDF.

(i) True Portuguese word score vector using gVis vectors and traditional TF–IDF.

Figure 8.4 (cont.): An example video and its 16 word score vectors, each represented by a word cloud.

(j) Predicted English word score vector using avgVis vectors and cluster-based TF–IDF.

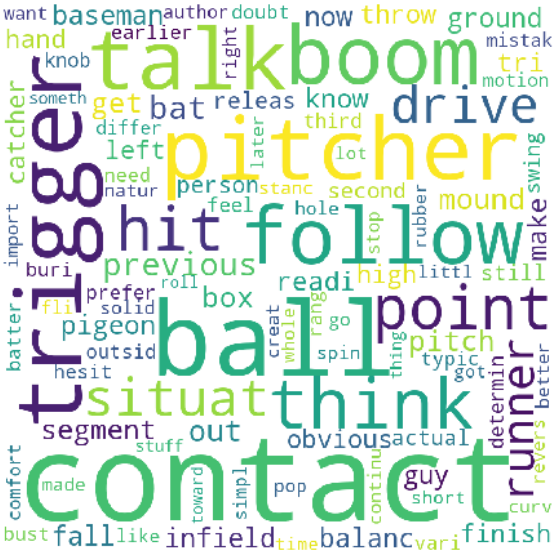
(k) True English word score vector using avgVis vectors and cluster-based TF–IDF.



(l) Predicted Portuguese word score vector using avgVis vectors and cluster-based TF–IDF.

(m) True Portuguese word score vector using avgVis vectors and cluster-based TF–IDF.

Figure 8.4 (cont.): An example video and its 16 word score vectors, each represented by a word cloud.



(n) Predicted English word score vector using gVis vectors and cluster-based TF–IDF.

(o) True English word score vector using gVis vectors and cluster-based TF–IDF.

(p) Predicted Portuguese word score vector using gVis vectors and cluster-based TF–IDF.

(q) True Portuguese word score vector using gVis vectors and cluster-based TF–IDF.

Figure 8.4 (cont.): An example video and its 16 word score vectors, each represented by a word cloud.

Chapter 9: Optimization

A number of optimization techniques were explored, and a few were employed to decrease the computational complexity of the problem. The first optimization method that is used is using trees for the KNN search. We also make use of multi-threading when computing the gVis vectors. Lastly, we explore the potential for dimensionality reduction using PCA, which will in turn reduce computational complexity.

9.1 Nearest Neighbor Trees

The sklearn package for KNN comes equipped with 2 tree methods for reducing computational complexity: Ball Tree and k-dimensional (k-d) Tree [9]. Tree algorithms are a useful way to speed up the computation of nearest neighbors, by reducing the number of distances that need to be calculated. The trivial method of computing the nearest neighbors for a test point requires computing the distance from the point in the space to every other point in the space. To compute the nearest neighbors in a d dimensional space with n training points using this brute force method requires $\mathcal{O}(dn)$ operations. To reduce this complexity, the space can be divided up into subspaces by using trees. The nearest neighbors of a test point can then be calculated more quickly by first observing which subspace the test point lies in. Using a k-d or ball tree reduces the computational complexity of finding nearest

neighbors to $\mathcal{O}(log(n))$. Using ball trees, the space is divided into a number of nested hyper-spheres. The ball tree structure works better for higher dimensional data compared to the k-d tree. When using a k-d tree, the space is split into smaller subspaces along alternating axes. The k-d tree works better for lower dimensional data. It is important to note that when using trees to optimize nearest neighbor search, the exact neighbors are not guaranteed. By default, the sklearn package automatically chooses which optimization method to use given the data. The code used for this thesis uses the default sklearn parameters for optimization.

9.2 Multithreading

Multithreading is useful because many tasks can be completed simultaneously. For this thesis, we implement multi-threading when calculating the gVis vectors. As mentioned previously, having to compute the color histograms for every frame of the video makes the gVis vectors computationally intensive. Multithreading provides a significant and necessary speed up when generating the gVis vectors. Using multithreading, each CPU core available computes a gVis vector independently of the other cores. Multithreading may also be able to be applied to other areas of the code to improve performance.

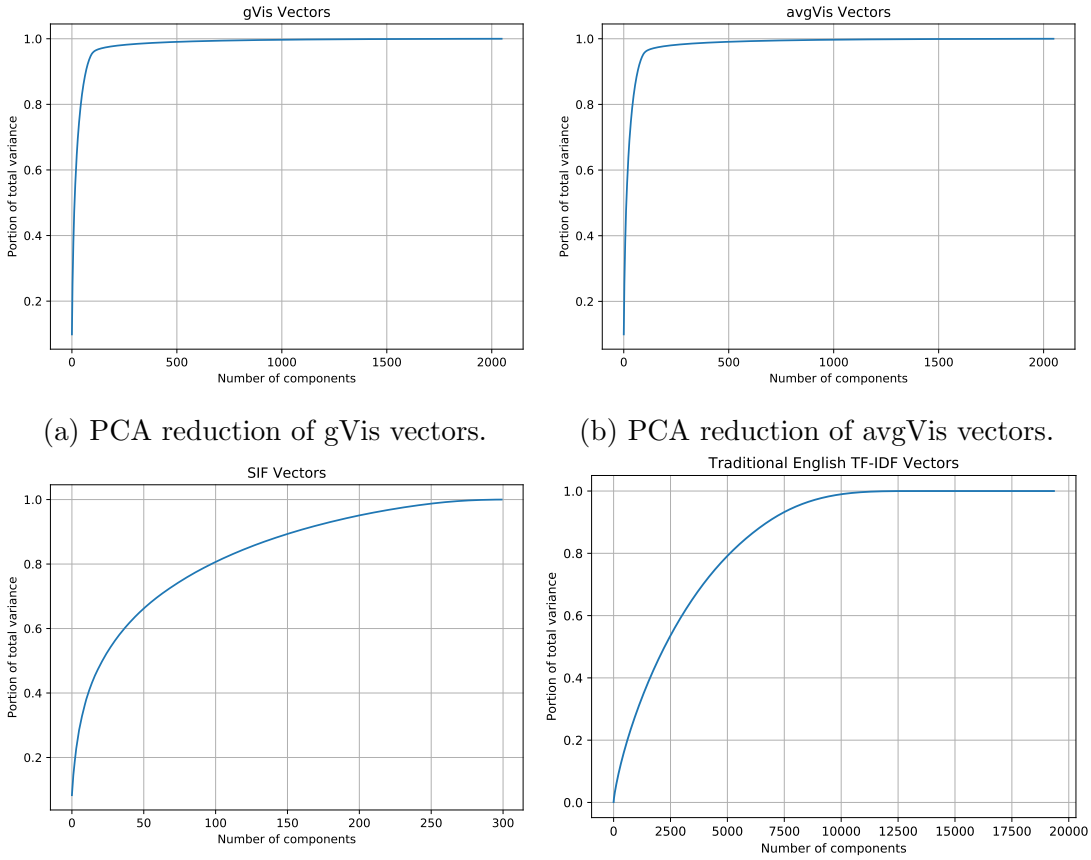
9.3 Dimensionality Reduction

Often times high dimensional data can be approximated using a lower number of dimensions. To get the best representation in a lower number of dimensions, data should be projected onto orthogonal principal components. Principal components can be calculated for a matrix of data, X, where each row of X corresponds to a

sample vector from the training set. Principal components are calculated from eigen decomposition of the covariance matrix $\mathbf{R}_{XX}$:

$$\mathbf{R}_{XX} = Q\Lambda Q^T \tag{9.1}$$

Assuming the eigenvalues in Q are arranged from largest to smallest, the n^{th} principal component is the n^{th} column of Q. Projecting the data in X onto the first n principal components gives the maximizes variance for the data in n dimensional space. The portion of total variance can then be plotted as a function of the number of principal components, as shown in Fig. 9.1. It is shown in Fig. 9.1 that much of the variance for the visual feature vectors (both avgVis and gVis) can be preserved even in dimensions much lower than the 2048-dimensional space they are currently in. The SIF vectors on the other hand, require a significant portion of their 300 components to account for most of the variance. The TF–IDF vectors represented in Fig. 9.1 were computed in English for the training videos using traditional TF–IDF scoring. The plot shows that the TF–IDF vectors could be reduced in dimensionality by around 50 percent using PCA. Using PCA to reduce the dimensionality of any feature vectors would reduce computational complexity as well.



(a) PCA reduction of gVis vectors.

(b) PCA reduction of avgVis vectors.

(c) PCA reduction of SIF vectors.

(d) PCA reduction of TF–IDF vectors.

Figure 9.1: Variance vs number of principal components for different feature vectors.

Chapter 10: Conclusions, Contributions, and Potential Future Work

In Chap. 1, we introduced the problem of multimodal data processing. We saw that predicting context vectors that assign an importance score for each word given visual information could be used in a number of applications where multimodal data can be collected. Additionally, we saw that neural networks are a common tool in multimodal processing today, but that other methods for processing multimodal data existed prior. Chapter 3 introduced the multimodal How2 dataset, which was used throughout this thesis. The major information used from the How2 dataset was the visual feature vectors and the transcriptions for each video.

In Chap. 4, we explored methods for representing the visual information of a video. We first explored a trivial method of averaging feature vectors to generate avgVis vectors. Then, a novel method was explored to generate gVis vectors using scene detection and weighted averages. A method for unsupervised clustering of visual features using FINCH was presented. Chapter 5 showed how the information from the transcript of videos can be intuitively represented using TF–IDF vectors. We introduced a novel modification to TF–IDF scoring that uses information from clusters. This chapter also explored methods for preprocessing and cleaning text so that the text is more usable.

Chapter 6 introduced how a nearest neighbors approach can be used to approximate the TF–IDF score for a video, given the TF–IDF scores of its neighbors. The neighbors of the video could be calculated in the visual feature space. We also introduced a purity metric that measured how well a video fit into the training space.

Chapter 7 explored methods to visualize the relationship between the linguistic information and the visual features for a set of videos. To create this visualization, this chapter first explored a smooth inverse frequency weighting to generate embedding vectors from a weighted average of word embedding vectors.

The results chapter, Chap. 8, gave a demonstration of the visual feature vectors that can be predicted for a video, using the proposed pipeline. We showed there are many methods to generate word score vectors, and all provide reasonable context for the video. Lastly, Chap. 9 explored some techniques that can be used to optimize the algorithms in the pipeline. This chapter also showed using principal component analysis, how many dimensions are needed to capture variance for the different feature vector representations used in this thesis.

10.1 Contributions

This thesis provided a number of contributions and insights into the problem of processing multimodal data. Specifically, this thesis looked at joint visual and linguistic information extractracted from videos. The visual information used came from the frames of the videos, and the linguistic information came from the subtitles of what was said in the videos. A list of important novel contributions includes:

- Introduced a method to summarize multiple visual feature vectors of a video using Gaussian weighted average scene vectors.

- Introduced a modification to the widely used TF–IDF word scoring method which exploits information stored in clusters.
- Developed a full pipeline for predicting a TF–IDF score vector using a summary of the visual features of a video.
- Introduced a purity score to represent how well a new point fits into the training dataset.
- Provided a method to visualize the relationship between the visual feature space and linguistic information space.

10.2 Potential Future Work

One possibility for future work would be to use a neural network to map visual features to SIF feature representations. Preliminary attempts to train a neural network for performing that task have shown that this problem may be ill-conditioned. Furthermore, the visualizations provided in Sect. 7.2 show that for many videos there may not exist a reliable mapping from visual feature vectors to text vectors. In the future, another approach would be to establish a method for determining if a visual feature vector is likely to provide a strong mapping to the word space, similar to the previously discussed purity score. This would allow for a measure of confidence in the predicted word score vectors. The confidence measure could then be used to select examples for training a neural network in a semi-supervised fashion.

Another avenue for future work is to modify the way that the test videos neighbors' word score vectors are averaged. The current method is to take an unweighted average of the k nearest neighbors. An alternative method could be to take a weighted average

with weights based on the distance between the training sample and the test point. Different weighting functions could be explored to determine what type of weighting gives the best results.

10.3 Final Thoughts

Multimodal data processing is a useful tool that can applied to a variety of problems. Although neural networks can provide good performance for mapping nonlinear transformations, simple and intuitive methods should not be overlooked when it comes to multimodal processing. In this thesis, we proposed novel modifications to a number of existing methods. We then combined these methods to create an overall pipeline that can be used to predict a feature vector for one modality, using feature vectors from a different modality. The proposed pipeline provides a novel, yet simple, approach to processing data in multiple modalities.

Appendix A: Discussion of Cosine Similarity

The cosine similarity between any two vectors, $v_1 \in \mathbb{R}^N$ and $v_2 \in \mathbb{R}^N$ is given by:

$$S(v_1, v_2) = \cos\theta = \frac{v_1 \cdot v_2}{\|v_1\| \, \|v_2\|} \tag{A.1}$$

where θ is the angle between the vectors, $\cdot$ is the dot product of the vectors, and $\|\cdot\|$ is the Euclidean norm. Cosine similarity is bounded between -1 and 1. The cosine distance between two vectors can be derived from the cosine similarity:

$$D(v_1, v_2) = 1 - S(v_1, v_2) \tag{A.2}$$

which is bounded between 0 and 2. Although the cosine distance is not a proper distance metric, it gives lower values when comparing similar vectors and higher values when comparing dissimilar vectors. Neither the cosine similarity nor the cosine distance is dependent on the magnitude of the vectors, only the angle between them. This means that non-zero vectors v_1 and v_2 can be L2-normaized without affecting the value of the cosine distance. Given that

$$\|v_1\|^2 = \|v_2\|^2 = 1 \tag{A.3}$$

the cosine distance can be simplified as

$$D(v_1, v_2) = (1 - v_1 \cdot v_2) \tag{A.4}$$

$$D(v_1, v_2) = (1 - v_1^T v_2) \tag{A.5}$$

$$D(v_1, v_2) = \frac{1}{2}(2 - 2v_1^T v_2) \tag{A.6}$$

$$D(v_1, v_2) = \frac{1}{2}((v_1^T v_1 + v_2^T v_2) - v_1^T v_2 - v_2^T v_1) \tag{A.7}$$

$$D(v_1, v_2) = \frac{1}{2}(v_1^T v_1 - v_1^T v_2 - v_2^T v_1 + v_2^T v_2) \tag{A.8}$$

$$D(v_1, v_2) = \frac{1}{2}(v_1^T (v_1 - v_2) - v_2^T (v_1 - v_2)) \tag{A.9}$$

$$D(v_1, v_2) = \frac{1}{2}((v_1^T - v_2^T)(v_1 - v_2)) \tag{A.10}$$

$$D(v_1, v_2) = \frac{1}{2}((v_1 - v_2)^T (v_1 - v_2)) \tag{A.11}$$

$$D(v_1, v_2) = \frac{1}{2} \left\| v_1 - v_2 \right\|^2 \tag{A.12}$$

which is half of the squared Euclidean distance between v_1 and v_2. This means that for ordering the distances between unit vectors, the cosine distance and the Euclidean distance will give the same result. For this thesis, we care only about the relative distances between word embeddings, which is why we can normalize the embedding vectors and use Euclidean distance, rather than cosine distance.

Appendix B: Videos Removed from How2 dataset

The following 49 videos were excluded from this thesis because the visual features were not available for download from the How2 download site. The 48 videos removed from the testing set:

- -g45vqccdzI
- FZHlsBRa8o0
- FZLxEwsoc1c
- FZNuNG9UBnw
- FZbyRzy4huk
- FZmRZ-GKlYo
- FzOQMA-CVPc
- FzmL8SL6Bow
- G06Irzcwxiw
- G0MjvzT_UqM
- G1QiXuldOxM

- G1hb5HugzVk
- G1lNlhjWC1I
- G2-GOjEMp4c
- G23JltC2N8g
- G25fic3QxDk
- G2JE_BEFVmE
- G2M7s9uE7f0
- G2T2WCuU39Y
- G2VAlFdgof4
- G3CyVk6dizw
- G3GcPpidwxk
- G3IJAoK0uSE
- G3RvsnzQrXQ
- G3g0-BeFN3c
- G42xKICVj9U
- fzDHRCKr7wU
- fzNnTgHa1lM
- fzrJBu2qsM8

- g08M_BtZcVU
- g0fCKNtiaoU
- g0fgci8L_rc
- g0t4Wz5qsT8
- g0uGWjwdOIw
- g13Jo_kFliM
- g1YJhiQ06o0
- g1fv1N7DZG4
- g25IORxoyXE
- g2iFC1st7zQ
- g2nvBjp0loQ
- g2v-M6EXcUE
- g3Cc_1-V31U
- g3PBeTb1TCw
- g3ZluRtRXVc
- g3jQ5ecjGz8

There is 1 video removed from the training set due to incomplete visual feature information:

- cnEIBUIiGik

One additional video was removed from the training set because it had no useful subtitle information:

- Ec9Qqd5EtTM

Appendix C: Code

C.1 Keyframe Detection

The Python function that is used to detect keyframes is shown below.

```
import cv2
import os
import numpy as np

def get_key_frames(vid_file_name, vid_id, use_dir=None):
    """ Returns a list of key frame indicies given a video filename.
        use_dir to pull saved histograms instead of generating them """
    # Threshold
    t_roll = 0.1
    MAX_IMG_SIZE = 100
    color_convert = cv2.COLOR_BGR2LAB  # frames loaded as BGR
    n_bin = 5
    # number of bins for 3 channels
    n_bins = np.array([1, n_bin, n_bin])

    bin_size = 256 / n_bins
    cam = cv2.VideoCapture(vid_file_name)
    # Get video scale
    width = int(cam.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(cam.get(cv2.CAP_PROP_FRAME_HEIGHT))
    scale = MAX_IMG_SIZE / max(width, height)
    frame_size = (round(scale * width), round(scale * height))
    # n_frames may be a bit off
    n_frames = int(cam.get(cv2.CAP_PROP_FRAME_COUNT))

    if not os.path.exists(f"{use_dir}/{vid_id}.npy"):
        # Create the color histogram, 1 row per frame, 1 col per bin
```

```
        # (extra row in case frame count is off)
        color_his_arr = np.zeros((n_frames + 1, np.product(n_bins)))
        # Get first frame
        ret, frame = cam.read()
        # Pre-allocate these for speed
        bin_num_rgb = np.zeros((3, np.product(frame_size)))
        bin_num_lin = np.zeros((np.product(frame_size)))
        fn = 0
        tic = time.time()
        while True:
            # Resize and convert colors
            frame = cv2.resize(frame, frame_size, fx=0, fy=0,
                               interpolation=cv2.INTER_CUBIC)
            frame = cv2.cvtColor(frame, color_convert)
            # Rasterize frame pixels and convert colors to bin numbers
            # bin_num_rgb is 1 col per pixel, 1 row per channel
            bin_num_rgb[:, :] = np.floor(
                frame.reshape((3, -1)) / bin_size[:, None])
            # Convert 3d hist to linear histogram
            bin_num_lin[:] = np.ravel_multi_index(
                bin_num_rgb.astype(int), n_bins.astype(int))
            color_his_arr[fn, :] = np.bincount(
                bin_num_lin.astype(int), minlength=np.product(n_bins))

            ret, frame = cam.read()
            fn = fn + 1
            if not ret:
                break
        # In case n_frames was a bit wrong
        n_frames = fn
        color_his_arr = color_his_arr[:n_frames, :]
    else:
        color_his_arr = np.load(f"{use_dir}/{vid_id}.npy")
        n_frames = color_his_arr.shape[0]

    """ Chi squared rolling """
    rolling_dist = np.empty(n_frames)
    rolling_changes = []
    ref_frame_ind = 0
    for i in range(1, n_frames):
        h1 = color_his_arr[ref_frame_ind, :]
        h2 = color_his_arr[i, :]
        # dont use bin if den is 0
```

```
        inds = np.where(h1 + h2 != 0)
        h1 = (h1 / sum(h1))[inds]
        h2 = (h2 / sum(h2))[inds]
        rolling_dist[i] = 0.5 * np.sum((h1 - h2)**2 / (h1 + h2))
        # see if we are different enough from the ref frame
        if rolling_dist[i] > t_roll:
            ref_frame_ind = i
            rolling_changes.append(i)
    return rolling_changes
```

C.2 Word processing

The code that is used for preprocessing the words for the TF–IDF vectors is shown below. The subtitles for each video are passed into `preprocess_func()` as a single string. The packages required for this code are [11] [15].

```
import snowballstemmer
import stop_words
import string

""" Preprocess the words """
# Make the stemmer
stemmer = snowballstemmer.stemmer(lang)
# Remove punctuation
# (include extra punctuations)
punctuation = string.punctuation
punctuation = punctuation.replace("%", "")
punctuation = punctuation.replace("$", "")
punctuation = punctuation.replace('\u0027', "")
punctuation = punctuation.replace('\u0060', "")
punctuation = punctuation + '\u201c\u201d\u2018\u2026\u2013\u2014' + '0123456789'
punc_trans = str.maketrans(punctuation, ' ' * len(punctuation))
# Characters that might be used as apostrophe '
apostrophe_trans = str.maketrans('', '', "\u2019\u02BC\u0027\u0060")
# EM, EN and hyphen should all be converted to spaces
# Take out all the stop words (remove punctuation from stop words since we
# remove punctuation from other words first)
stopwords = stop_words.get_stop_words(lang)
stopwords = " ".join(stopwords).translate(
```

```
    punc_trans).translate(apostrophe_trans).lower().split()


def preprocess_func(words):
    """ runs the preprocessing function for the given string of words and
        returns the processed words """
    # replace punctuation with spaces
    words = words.translate(punc_trans)
    # remove apostrophes
    words = words.translate(apostrophe_trans).lower().split()
    # remove stopwords
    words = [word for word in words if word not in stopwords]
    # stem the words (only used for TF-IDF vectors, not SIF embeddings)
    words = stemmer.stemWords(words)
    # convert back to one string
    words = " ".join(words)
    return words
```

Appendix D: Stop Words

D.1 English

The following stop words were used for English, as provided by the `stop_words` package in Python [15]:

a, about, above, after, again, against, all, am, an, and, any, are, aren't, as, at, be, because, been, before, being, below, between, both, but, by, can't, cannot, could, couldn't, did, didn't, do, does, doesn't, doing, don't, down, during, each, few, for, from, further, had, hadn't, has, hasn't, have, haven't, having, he, he'd, he'll, he's, her, here, here's, hers, herself, him, himself, his, how, how's, i, i'd, i'll, i'm, i've, if, in, into, is, isn't, it, it's, its, itself, let's, me, more, most, mustn't, my, myself, no, nor, not, of, off, on, once, only, or, other, ought, our, ours, ourselves, out, over, own, same, shan't, she, she'd, she'll, she's, should, shouldn't, so, some, such, than, that, that's, the, their, theirs, them, themselves, then, there, there's, these, they, they'd, they'll, they're, they've, this, those, through, to, too, under, until, up, very, was, wasn't, we, we'd, we'll, we're, we've, were, weren't, what, what's, when, when's, where, where's, which, while, who, who's, whom, why, why's, with, won't, would, wouldn't, you, you'd, you'll, you're, you've, your, yours, yourself, yourselves

D.2 Portuguese

The following stop words were used for Portuguese, as provided by the `stop_words` package in Python [15]:

a, ao, aos, aquela, aquelas, aquele, aqueles, aquilo, as, até, com, como, da, das, de, dela, delas, dele, deles, depois, do, dos, e, ela, elas, ele, eles, em, entre, era, eram, essa, essas, esse, esses, esta, estamos, estas, estava, estavam, este, esteja, estejam, estejamos, estes, esteve, estive, estivemos, estiver, estivera, estiveram, estiverem, estivermos, estivesse, estivessem, estivéramos, estivéssemos, estou, está, estávamos, estão, eu, foi, fomos, for, fora, foram, forem, formos, fosse, fossem, fui, fôramos, fôssemos, haja, hajam, hajamos, havemos, hei, houve, houvemos, houver, houvera, houveram, houverei, houverem, houveremos, houveria, houveriam, houvermos, houverá, houverão, houveríamos, houvesse, houvessem, houvéramos, houvéssemos, há, hão, isso, isto, já, lhe, lhes, mais, mas, me, mesmo, meu, meus, minha, minhas, muito, na, nas, nem, no, nos, nossa, nossas, nosso, nossos, num, numa, não, nós, o, os, ou, para, pela, pelas, pelo, pelos, por, qual, quando, que, quem, se, seja, sejam, sejamos, sem, serei, seremos, seria, seriam, será, serão, seríamos, seu, seus, somos, sou, sua, suas, são, só, também, te, tem, temos, tenha, tenham, tenhamos, tenho, terei, teremos, teria, teriam, terá, terão, teríamos, teu, teus, teve, tinha, tinham, tive, tivemos, tiver, tivera, tiveram, tiverem, tivermos, tivesse, tivessem, tivéramos, tivéssemos, tu, tua, tuas, tém, tínhamos, um, uma, você, vocês, vos, à, às, éramos

Bibliography

[1] Antonios Anastasopoulos, Shankar Kumar, and Hank Liao. Neural language modeling with visual features. *arXiv preprint arXiv:1903.02930*, 2019.

[2] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. A simple but tough-to-beat baseline for sentence embeddings. In *5th International Conference on Learning Representations*, 2017.

[3] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

[4] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[5] Kensho Hara, Hirokatsu Kataoka, and Yutaka Satoh. Can spatiotemporal 3D CNNs retrace the history of 2D CNNs and ImageNet? In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 6546–6555, 2018.

[6] Leland McInnes, John Healy, and James Melville. UMAP: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[7] Yasuhide Mori, Hironobu Takahashi, and Ryuichi Oka. Automatic word assignment to images based on dividing image and vector quantization. In *Proceedings of Computer-Assisted Information Retrieval*. CID, 2000.

[8] Andreas Mueller. Wordcloud for python. `https://amueller.github.io/word_cloud/`, 2020.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[10] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global vectors for word representation. In *Empirical Methods in Natural Language Processing*, pages 1532–1543, 2014.

[11] Martin Porter and Richard Boulton. Snowball. `https://snowballstem.org/`, 2021.

[12] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, et al. The kaldi speech recognition toolkit. In *IEEE 2011 workshop on automatic speech recognition and understanding*. IEEE Signal Processing Society, 2011.

[13] Ramon Sanabria, Ozan Caglayan, Shruti Palaskar, Desmond Elliott, Loïc Barrault, Lucia Specia, and Florian Metze. How2: a large-scale dataset for multimodal language understanding. In *Proceedings of the Workshop on Visually Grounded Interaction and Language*. NeurIPS, 2018.

[14] Saquib Sarfraz, Vivek Sharma, and Rainer Stiefelhagen. Efficient parameter-free clustering using first neighbor relations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8934–8943, 2019.

[15] Alireza Savand and François. Python stop words. `https://github.com/Alir3z4/python-stop-words`, 2018.

[16] Suramya Tomar. Converting video formats with FFmpeg. *Linux Journal*, 2006(146):10, 2006.

[17] Laurens van der Maaten and Geoffrey Hinton. Viualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 11 2008.

[18] John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. From paraphrase database to compositional paraphrase model and back. *Transactions of the Association for Computational Linguistics*, 3:345–358, 2015.

[19] Shoou-I Yu, Lu Jiang, and Alexander Hauptmann. Instructional videos for unsupervised harvesting and learning of action examples. In *Proceedings of the 22nd ACM International Conference on Multimedia*, 2014.

[20] Zhang Yun-tao, Gong Ling, and Wang Yong-cheng. An improved tf-idf approach for text classification. *Journal of Zhejiang University-Science A*, 6(1):49–55, 2005.